우리식조작체계문고

통합개발환경
Eclipse



교육성교육정보쎈터
주체97(2008)년

차 례

머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《프로그람을 개발하는데서 기본은 우리 식의 프로그람을 개발하는것입니다. 우리는 우리 식의 프로그람을 개발하는 방향으로 나가야 합니다.》(《**김정일**선집》 제15권, 196페지)

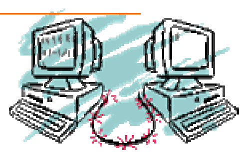
위대한 장군님의 원대한 구상에 따라 우리식 조작체계의 개발이 힘있게 추진되고있는 오늘 여기에 필요한 프로그람개발도구를 갖추는것이 중요한 문제로 나서고있다. 우리식 조작체계에 기초하여 인민경제 여러 부문에서 필요되는 응용프로그람들을 충분히 보장하려면 여러가지 기능을 유연하게 구성할수 있는 통합개발환경을 가진 프로그람개발도구가 있어야 한다.

Eclipse는 이와 같은 현실적요구를 만족시킬수 있는 공개원천통합개발환경으로서 쏘프트웨어개발에 종사하는 전문가, 기술자들의 높은 평가를 받고있으며 널리 리용되고있다.

우리는 Eclipse를 조선어환경에 적응시키고 기능을 전용화한 우리 식의 조작체계개발도구로 전환시키는데서 초보적인 성과를 거두었다.

이 책에서는 통합개발환경 Eclipse에 대한 일반적지식과 Eclipse를 리용한 Java프로그람작성, Web응용프로그람개발을 비롯한 여러가지 개발방법들을 서술하고있다.

이 책은 이미 최신개발도구를 습득하고 우리 식의 쏘프트웨어개발에 참가하여 지혜와 정열을 다 바치고있는 이 부문 전문가, 기술자들에게 참고가 되는것은 물론이고 앞날의 주인공들인 우리의 대학생들에게도 좋은 길동무가 되리라고 본다.



제1장. Eclipse의 개요

제1절. Eclipse란 무엇인가

Eclipse는 Eclipse Foundation라는 단체가 개발하고있는 개발도구를 위한 가동기반이다. Eclipse는 여러개의 부분프로젝트를 가지고있고 Eclipse의 표준개발묶음(Eclipse SDK)에는 JDT(Java Development Tools: Java개발도구)라고 하는 Java통합개발환경(IDE)도 포함되여있다.

Eclipse의 기본목적의 하나는 공개원천으로 상업수준의 높은 기능의 개발환경을 제공하는것이다. 무상으로 제공되는 기본적인 Java개발환경만 가지고도 실용성이 충분하며 직업적개발전문가들이 즐겨 리용하고있다. 사실상 Eclipse자체도 Eclipse로 개발되고있다. Eclipse본체는 공개원천이고 Common Public License(CPL) 또는 Eclipse Public License(EPL)라고 하는 상업적리용에서 허용도가 높은 허가를 적용할수 있다.

Eclipse의 특징을 세가지로 볼수 있다.

첫째로, 고속사용자대면부이다.

Eclipse의 표준도형사용자대면부에는 SWT(Standard Widget Toolkit: 표준부품도구묶음)라고 하는 Swing보다 가볍고 속도가 빠른 일반도형서고가 리용되므로 Swing에 의존하는 응용제품보다 더 좋은 조작성을 실현하고있다.

둘째로, 높은 기능을 갖춘 Java개발환경이다.

Ant, JUnit 등 Java의 표준적인 개발도구를 지원할뿐아니라 건기능, 오유수정기능, CVS(Concurrent Versions System: 병행판본체계) 등의 판본관리체계의 지원을 비롯하여 Java통합개발환경으로서의 기능을 기본적으로 갖추고있다.

셋째로, 끼워넣기(plug-in)에 의한 확장성을 가지고있다.

Eclipse의 모든 기능은 다 끼워넣기에 의하여 실현된다. 그러므로 필요한 기능을 가지는 끼워넣기를 조합함으로써 자기가 바라는 개발환경을 구축할수 있다. 표준적인 구성을 가진 Eclipse에는 끼워넣기를 개발하기 위한 PDE(Plug-in development environment: 끼워넣기개발환경)가 갖추어져 있으므로 끼워넣기개발도 Eclipse로 할수 있다. Eclipse는 설계형태를 제기한 Eric Gamma를 비롯하여 세계적으로 유명한 프로그람개발자들이 개발리용하고있는 도구이다. 즉 Eclipse는 단순한 초보자용도구가 아니라 전문가들이 실제로 개발현장에서 쓰면서 발전시켜온 특색있는 도구이다.

1.1.1. Eclipse의 력사

1999년에 OTI(Object Technology International:객체기술국제협의회)와 IBM이 공동으로 공개가동기반인 Eclipse개발을 시작하였다. Eclipse는 개발 첫시기부터 40명의 전문개발자들에 의하여 본격적인 개발이 진행되였으며 2001년에는 정식으로 공개원천화되여 Eclipse련맹이 설립되였다.

Eclipse1.0은 2001년 10월에 발표되고 11월에는 eclipse.org거점이 생겨 내리적재가 가능하게 되였다. 2002년 6월에는 Eclipse2.0이 발표되였는데 Eclipse1.0에 비하여 조작성이 개선되고 끼워넣기의 설치와 갱신기능이 추가되였다. 그후 2003년 3월에 2.1이 발표되고 2.1.1을 거쳐 2.1.3까지 발표되였다. 2004년 6월에는 Eclipse3.0이 발표되였다. Eclipse3.0에서는 사용자들의 요청을 받아들여 사용자대면부가 많이 개선되였다. 또한 통합개발환경의 가동기반으로서뿐아니라 다기능 의뢰기응용프로그람의 가동기반으로서의 기능도 추가되였다. Eclipse3.0은 리용범위를 도구에 극한시키지 않고 실행환경까지 확장한 가동기반이라고 말할수 있다.

표 1-1. Eclipse의 력사

1999년	4월	OTI/IBM의 내부에서 Eclipse개발시작
2000년	6월	Eclipse Technique Preview를 발표
2001년	3월	첫 Eclipse거점 (http://www.eclipsecorner.org/)개설
	6월	Eclipse0.9 발표
	10월	Eclipse1.0 발표
	11월	IBM이 Eclipse원천코드를 기증하여 IBM외 9개 회사가 주도성원이 되여 Eclipse련맹이 설립 Eclipse거점(http://www.eclipse.org/)이 개설
2002년	3월	3개 회사가 새로 주도성원으로 참가
	6월	6개 회사가 주도성원으로 참가, Eclipse2.0발표
	8월	Eclipse2.0.1발표
	9월	2.0용국제화모듈발표, Oracle이 주도성원으로 참가
	11월	Eclipse2.0.2발표
	12월	OMG(Object Management Group)가 주도성원으로 참가
2003년	3월	Eclipse2.1발표, Eclipse3.0개발계획을 발표
	7월	Eclipse2.1.1발표, Eclipse2.1.x용국제화모듈발표, Intel이 주도성원으로 참가
2004년	2월	Eclipse가 독립단체로 재편성, 첫 Eclipse국제토론회 개최, Eclipse Plugin Central (http://www.eclipsepluginCentral.com /)개설
	6월	Eclipse3.0발표
	9월	Eclipse3.0.1발표, Eclipse3.0.1용국제화모듈발표

1.1.2. Eclipse프로젝트

Eclipse프로젝트개발은 공개원천독립단체인 Eclipse기금(Eclipse Foundation)에 의하여 추진되고있다. 이것은 처음에 Eclipse련맹이던것이 2004년에 첫 Eclipse국제토론회가 개최되면서 독립단체로 재결성되였다. 2004년 9월 eclipse.org안에는 5개 프로젝트가 존재하였다. 이 프로젝트들은 프로젝트관리위원회(PMC)에 의하여 관리되고있다.

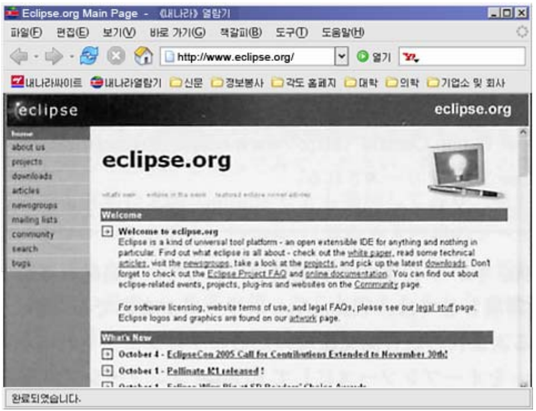


그림 1-1. eclipse.org거점

표 1-2. Eclipse프로젝트의 개괄

프로젝트	부분프로젝트	프로젝트의 목적
The Eclipse Project	Platform	도구의 통합을 위한 가동기반의 핵심기틀을 제공
	JDT - Java development tools	Java통합개발환경을 위한 끼워넣기를 제공
	PDE - Plug-in Development environment	끼워넣기개발을 위한 환경 제공
The Eclipse Tools Project	Hyades	쏘프트웨어의 품질보증을 위한 도구를 제공
	C/C++ IDE	C와 C++를 위한 통합개발환경 제공
	Graphical EditorFramework (GEF)Project	GUI편집기를 위한 기틀 제공
	EMF	XML에 의한 Java코드생성의 기틀을 제공
	COBOL	COBOL를 위한 통합개발환경 제공

프로젝트	부분프로젝트	프로젝트의 목적
	VE(Visual Editor)	Eclipse에 의한 GUI건립기를 제공하는 프로젝트이며 Swing/JFC와 SWT 건립기의 참조를 제공
	UML2	EMF를 기초로 하는 UML2.0의 메타모형을 제공하기 위한 프로젝트
The Eclipse Technology Project	Aspect Development Tools Project(AJDT)	AspectJ를 위한 개발도구 제공
	AspectJ	Java에 의한 Aspect지향프로그람작성을 위한 연구를 진행
	CME(Concern Manipulation Environment)	Aspect지향개발을 위한 통합가동기반을 제공
	ECESIS(EclipseCommunity Education Project)	Eclipse의 교육과 훈련용내용물을 개발, 추진
	Equinox	끼워넣기확장, 끼워넣기의 의존관계를 동적으로 변경하는 물림새를 조사
	Generative Model Transformer (GMT)	모형구동쏘프트웨어개발을 지원하는 도구계렬 정비
	Pollinate	Apache Beehive기틀을 위한 개발환경을 구축하는 프로젝트
	Koi	협력개발을 위한 도구 제공
	Stellation	Eclipse에 의한 쏘프트웨어구성관리에 대한 연구를 진행
	OMELET (Open Modeling Environment With Links for Extensions and Transformations)	각이한 모형의 표현, 모형변환, 모형변환기술을 통합하는 기틀을 제공
	XSD	XML Schema를 해석하고 조작하기 위한 서고를 제공
	WSVT (Web Service Validation Tools)	웨브봉사의 개발, 배포, 시험, 오유수정을 위한 검증도구를 제공
The Eclipse Web Tools Platform Project	Web Standard Tools	Eclipse를 기초로 하는 웨브응용프로그람의 공통개발환경을 제공
	J2EE Standard Tools	Eclipse를 기초로 하는 J2EE1.4기준의 응용프로그람의 공통개발환경을 제공

프로젝트	부분프로젝트	프로젝트의 목적
The Eclipse Test and Performance Tools Platform Project		Eclipse를 사용한 공개적인 시험도구의 가동기반을 제공, 현재 Hyades프로젝트를 재편성하여 프로젝트작성중
Business Intelligence And ReportingTools(BIRT)Project	Eclipse Report Designer(ERD)	보고서설계를 위한 탁상창작도구의 개발프로젝트
	Web BasedReport Designer(WRD)	자료조작을 위한 HTML기초의 보고서를 작성할수 있는 도구설계환경을 제공
	Eclipse Report Engine(ERE)	보고서의 생성 및 표시를 위한 응용프로그람개발을 진행할수 있게 한다
	Eclipse Charting Engine (ECE)	도표생성을 위한 부품을 제공
계획중에 있는 부분프로젝트	The Eclipse Communications Project	Eclipse를 사용한 통보전달 및 통신응용프로그람의 기능을 제공
	The Voice Tools Project	W3C표준을 기초로 하는 JSP/J2EE에서의 음성응용프로그람에 초점을 둔 프로젝트
	The Embedded Rich Client Platform (eRCP) Project	매몰형계렬의 다기능의뢰기환경을 제공하는 프로젝트

The Eclipse Project는 견고하고 많은 기능을 가진 시장류통제품수준의 쏘프트웨어 개발환경실현을 목적으로 하고있다. 이 프로젝트는 3개의 부분프로젝트로 갈라져있으며 각각 Eclipse의 핵심부분, Java통합개발환경(JDT), 끼워넣기개발환경(PDE)을 제공한다. 이 3개의 부분프로젝트는 Eclipse-SDK로서 묶어서 내리적재할수 있다.

The Eclipse Tools Project는 Eclipse를 위한 여러가지 도구들을 만들어내는 프로젝트이다. 이와 같은 프로젝트가 eclipse.org안에 있는것으로 하여 각이한 끼워넣기의 기능이 겹치지 않도록 조정할수 있다. 그리하여 The Eclipse Tools Project의 개발경험을 The Eclipse Project에 되돌려 도입하려는 목적도 있다.

The Eclipse Technology Project는 Eclipse의 발전에 기여하는 공개원천개발자와 연구사, 교육자를 위한 프로젝트이다. 이 프로젝트는 연구, 발명, 교육의 3가지 부

류로 나눈다.

The Eclipse Web Tools Platform Project는 Web응용 및 J2EE응용프로그람개발을 위한 가동기반을 제공하는 프로젝트이다. 특정한 쏘프트웨어개발자에 치우치지 않는 중립적이며 표준적인 도구에 대한 기본적인 가동기반을 제공한다.

The Eclipse Test and Performance Tools Platform Project는 시험도구의 가동기반을 제공하는 프로젝트이다. Eclipse Tools Project안의 Hyades프로젝트를 재편성하여 추진할 계획이다.

Business Intelligence and Reporting Tools(BIRT) Project는 업무지능화를 위한 도구를 제공하는 프로젝트이다. 2004년 10월에 정식으로 Eclipse의 프로젝트로 되였다.

Eclipse프로젝트개발은 공개원천방식으로 진행되고있으며 많은 개발자들이 개발에 참가하여 부단히 판본갱신이 진행되고있기때문에 프로젝트의 내용은 계속 변하고있다.

1.1.3. Common Public License와 Eclipse Public License

Eclipse2.x까지의 Eclipse프로젝트의 성과는 공용허가(CPL:Commmon Public License)라는 허가로 배포되여있다는것이다. CPL은 공개원천추진단체(OSI: Open Source Initiative, http://www.opensource.org/)가 인정하는 공개원천쏘프트웨어의 정의를 만족시키고있어 승인된것이다.

Eclipse3.x부터는 Eclipse Public License(EPL:http://www.eclipse.org/legal/epl-v10.html)라는 허가로 제공되도록 변경되였다. EPL도 OSI가 인정하는 공개원천허가이며 기본적으로는 CPL와 거의 같다. 주되는 차이점은 CPL이 IBM에 의하여 규정된것이라면 EPL는 Eclipse기금에 의하여 규정되였다는것과 특허에 관한 조항이 변경되여있다는것이다. Eclipse의 모듈속에는 CPL이나 EPL이외의 공개원천허가가 적용되여있는것도 있다. 또한 eclipse.org이외의 끼워넣기는 공개원천이 아닌것도 있기때문에 상업적목적에 리용하려는 경우에는 허가를 받았는가를 확인할 필요가 있다.

Eclipse의 상업적리용

Eclipse는 여러 기업들이 리용하고있다. IBM의 개발도구인 IBM Rational Application Developer 등은 Eclipse가동기반상에 만들어져있다. 그외에 자료기지의 개발도구나 ERP를 위한 개발도구들에도 Eclipse가동기반이 사용되고있다. Eclipse는 개발도구의 생태계(Ecosystem)를 제공하기때문에 매 개발업자들은 개발원가삭감, 제품의 호환성과 같은 리익을 향유할수 있다. 사용자들도 공통의 도형사용자대면부나 조작성으로하여 새로운 도구에도 재빨리 적응할수 있다는 우점이 있다.

1.1.4. Eclipse가동기반의 개요

Eclipse는 개발도구를 위한 확장가능한 가동기반이다. Eclipse가동기반은 각이한 환경에 대응하기 위하여 Java가상기계상에 구축되여있다.

Eclipse3.0이 대응하고있는 조작체계는 Linux, Windows, AIX, HP-UX, MacOS X이다. Eclipse는 Java실행환경(Java Runtime Environment:JRE)을 리용하고있기때문에 매개 조작체계환경에서의 Java실행환경이 필요하게 된다.

Eclipse가동기반의 특징중의 하나는 도형사용자대면부로 SWT를 리용하고있는것이다. SWT는 일반도형서고와 단추, 목록, 본문 등 부품의 집합체이다. Java도형사용자대면부환경에서 Swing이 있으면서도 SWT를 쓰는데는 그럴만한 리유가 있다. 그것은 복잡하고 비교적 뜨다고 평가되는 Swing부품에 비하여 실용적인 면에서 간단하고 고속으로 동작하는 화면부품을 제공하기 위해서이다. SWT가 제공하는 화면부품중에는 Java를 쓰지 않고 고유코드(native code)로 되여있는것이 있기때문에 순수한 Java(Pure Java)코드로 되여있다고는 말할수 없다. 때문에 SWT를 사용한 응용프로그람을 동작시키자면 Java실행환경외에 swt.dll의 2진파일을 설치할 필요가 있다. 그러나 많은 조작체계에 대응하는 swt.dll이 제공되여있기때문에 사실상 Eclipse는 가동기반에 무관계하게 쓸수 있다.

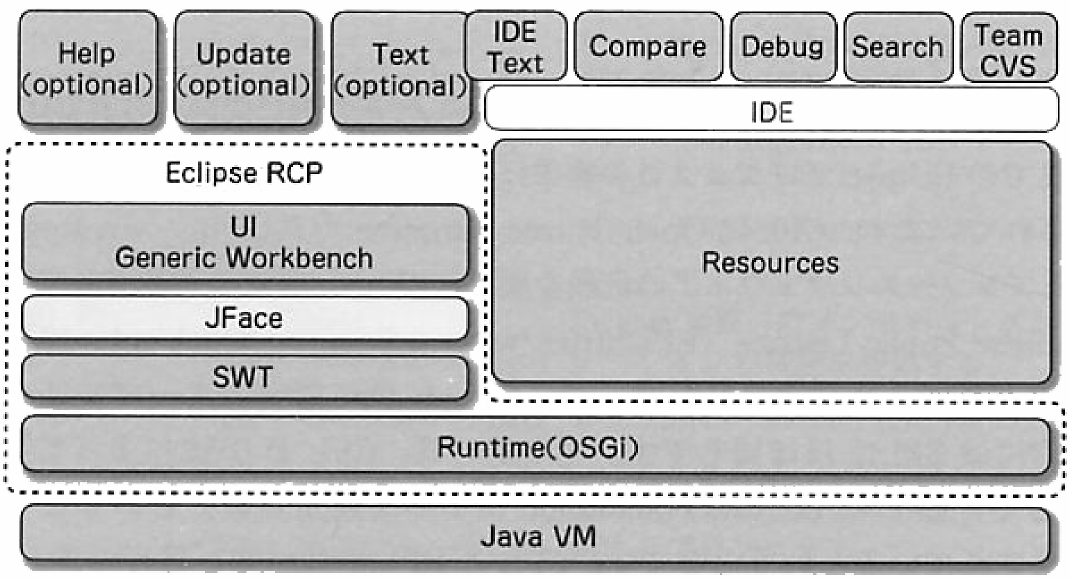


그림 1-2. Eclipse의 개요

Eclipse의 기능을 확장하기 위한 물림새가 끼워넣기이다. Eclipse의 화면에 표시되여있는 차림표나 편집기, 대화칸 등은 모두 끼워넣기에 의하여 이루어진것이다.

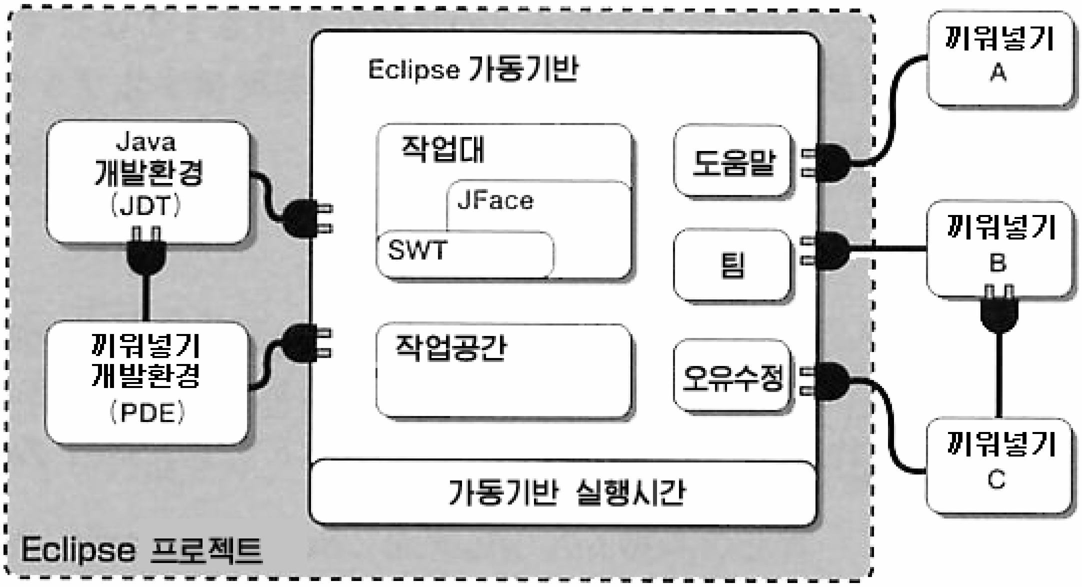


그림 1-3. Eclipse의 구성

그림 1-3에서 보는바와 같이 Eclipse프로젝트외의 끼워넣기를 추가하거나 자체로 개발한 끼워넣기를 추가할수도 있다. Java개발환경인 JDT나 끼워넣기개발환경인 PDE 자체도 Eclipse가동기반에 대한 끼워넣기이다. JDT를 리용하여 Java개발을 진행하는것 뿐이라면 Eclipse내부의 구성을 알 필요는 없다. 그러나 Eclipse의 우월성은 끼워넣기를 개발하거나 추가함으로써 자기가 바라는대로 환경을 구축할수 있다는데 있다. 그러자면 Eclipse의 내부구성을 알고있어야 한다.

- 끼워넣기의 구성방식

끼워넣기의 구성방식을 그림으로 표시하면 그림 1-4와 같다.

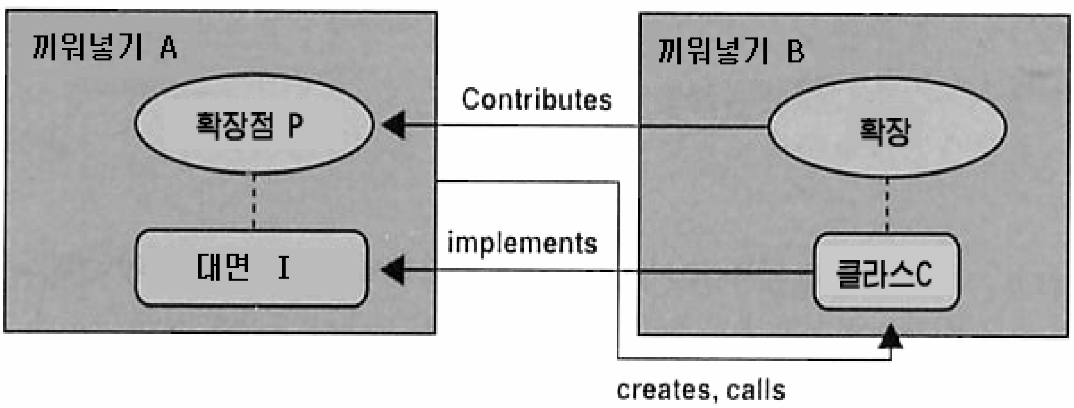


그림 1-4. 끼워넣기의 구성방식

그림 1-4에서는 끼워넣기B로부터 끼워넣기A의 기능을 리용하는 형식을 보여주고있다. 끼워넣기B의 모듈은 확장점에 대한 기여를 선언한다. 실지 동작흐름은 Eclipse가동기반으로부터 끼워넣기B가 호출되는 형식으로 된다. 이렇게 함으로써 원래부터 있던 끼워넣기A를 변경함이 없이 A의 기능을 리용한 끼워넣기B를 추가할수 있다. 사실 PDE의 끼워넣기는 JDT끼워넣기를 리용하여 만들어져있다. Eclipse가동기반은 이와 같은 끼워넣기의 물림새를 갖춤으로써 기존 끼워넣기를 리용한 기능확장을 실현하고있다.

Eclipse에서는 어떤 기능을 제공하는 끼워넣기의 총체를 특성정보(feature)라고 부른다. 그러므로 JDT는 Java개발을 위한 특성정보라고 말할수 있다.

- 작업대

작업대(work bench)란 Eclipse의 사용자대면부 즉 Eclipse창문안에서의 각종 보기의 총체이다. 작업대에는 SWT와 JFace가 포함된다.

SWT는 Eclipse화면을 작성하기 위한 도형사용자대면부도구함이다.

SWT의 특징은 간편성, 최소구성, 고속성을 가지며 조작체계에 의존하지 않는 응용프로그람대면부(API)와 고유한 화면부품을 만들수 있는것이다.

Swing을 쓰지 않고 고유한 화면부품을 쓰기때문에 매 조작체계의 표준대면부와 류사한 대면부형식을 실현하여 고속으로 동작시킬수 있다. SWT를 리용하여 사용자대면부의 공통파제부분을 실현하는 물림새를 제공하는것이 JFace이다. 대화(dialog), 나무보기창(tree view), 조수(wizard) 등의 비교적 조립도가 큰 화면부품이 준비되여있다. JFace의 응용프로그람대면부는 조작체계에 의존하지 않도록 만들어져있다.

- 작업공간

작업공간은 작업령역을 제공한다. 구체적으로 말하면 원천파일 등의 각종 자원이나 설정과 같은것을 보존하는 장소이다. Eclipse가 관리하는 각종 자원은 다 작업공간안에 보존된다.

- 도움말

말그대로 도움말을 제공한다. Eclipse의 도움말체계는 웨브를 기초로 하고있어 도움말의 내용을 HTML로 서술할수 있다. 또한 검색이나 책갈피(bookmark) 등 도움말을 지원하는 물림새도 제공하고있다. 도움말체계는 Java의 API문서표시 등에도 쓰인다. 표준적인 SDK의 도움말에는 Eclipse의 기능설명이나 개별지도서 등이 포함되여있다.

- 오유수정

오유수정은 프로그람의 실행, 멈춤점, 오유수정사용자대면부 등의 오유수정기능을 위한 물림새를 제공한다.

Eclipse3.0의 새로운 기능

Eclipse3.0은 Eclipse2.x에 비해 다음과 같은것들이 갱신되였다.

- 사용자대면부의 개선

기동할 때의 [Welcome]화면이 새롭게 만들어지고 개별지도서나 목록 등이 추가되였다. 또한 요약 및 묘리기능(tips and tricks)에 대한 설명 등이 추가되여있다. 그외에도 설계가 개선되고 보기의 겹침표시가 갱신되였다.

- SWT, 본문, 갱신, 조, 검색, Ant기능의 개선

Eclipse가 제공하는 가동기반이 가지는 기능과 응용프로그람대면부가 확장되였다. 례하면 과제접시(task tray)의 도형사용자대면부가 추가되였거나 끼워넣기의 갱신 등이 진행되였다.

- JDT, 오유수정, PDE기능의 개선

JDT와 Java가 가지고있는 개발기능이 강화되였다. 개편의 명칭을 변경할 때 설명문도 갱신되게 된것이라든가 추상문법나무보기창 (AST View: abstract syntax tree View)의 추가 등 개선된 점들이 많다.

1.1.5. 여러가지 끼워넣기

Eclipse에는 여러가지 끼워넣기들이 제공되여있다. Eclipse에서 리용할수 있는 끼워넣기는 너무 많아서 여기서는 다만 필요한 끼워넣기들이 있는 주요 거점을 소개하기로 한다. 주의해야 할것은 Eclipse3.0의 기술명세서가 Eclipse2.0과 완전한 호환이 진행되는것은 아니라는것이다. 따라서 Eclipse2.x용끼워넣기가 Eclipse3.0에서 그대로 동작하지 않을수 있다. 끼워넣기를 리용할 때에는 반드시 대응되는 Eclipse의 판본을 확인하여야 한다.

- Eclipse plugins http://www.eclipse-plugins.info

Urbaan이 관리하는 Eclipse용끼워넣기가 들어있는 거점이다. 650개이상의 끼워넣기가 분야별로 등록되여있다.

- Eclipse plugin Central http://www.eclipseplugincentral.com

Eclipse용끼워넣기의 류통을 위한 시장거점이다.

- SourceForge http://sourceforge.net

Eclipse용끼워넣기뿐아니라 공개원천프로젝트들에 대한 거점이다.

제2절. 설 치

여기서는 Eclipse를 Linux환경에 설치하는 방법을 설명한다. 우리 식 조작체계를 리용할 때에는 Ecipse3.0을 우리 식 조작체계용으로 개작한 《어깨동무》1.2판(《Companion》)을 쓴다. 일부 끼워넣기는 Windows판본에만 있다.

- 필요한 파일

Eclipse를 동작시키려면 다음과 같은 쏘프트웨어들이 필요하다.

· JDK v1.6.0의 JRE 또는 SDK(Software development kit)

· Eclipse 3.0 SDK

- J2SE의 설치

Eclipse3.0의 실행을 하자면 JDK v1.6.0의 Java실행환경이 필요하다. 그러나 실지로 Java개발을 진행하려면 Java실행환경만이 아니라 SDK도 있으면 편리하다. 여기서는 Jdk-6_0-ea-bin-b13-linux-i586-18_nov_2004-rpm.bin의 SDK를 설치한다.

- Eclipse-SDK의 설치

다음으로 Eclipse-SDK를 설치한다. 설치는 압축된 파일을 풀기만하면 된다. 여기서는 《/kut/eclipse》에 Eclipse본체파일이 설치된것으로 약속한다. 우리 식 조작체계에서 《어깨동무》1.2판을 리용할 때에는 《/Companero-1.2》에 설치된다.

- **언어꾸러미**(Language Pack)의 설치

Eclipse3.0에는 "Language Pack"라고 하는 여러나라 언어대응을 위한 모듈이 제공되여있다. Eclipse-SDK자체는 영어에만 대응되여있으나 언어꾸러미를 도입함으로써 지역화정보에 따라서 각 민족어로 차림표나 도움말 등이 표시되게 된다. 언어꾸러미의 설치도 Eclipse와 마찬가지로 압축파일을 풀기만하면 된다. 여기서 주의할 점은 언어꾸러미를 설치하기 전에는 Eclipse를 기동하지 말아야 한다는것이다. 언어꾸러미의 도입전에 Eclipse를 기동하면 영어방식에서의 설정파일이 작성되여 언어꾸러미를 적용한 후에도 설정의 일부가 영어방식으로 남아있으면서 일부 차림표나 서체의 설정이 영어로 되여있는 경우가 있다.

- 기동지름아이콘의 작성

《/kut/eclipse/eclipse(또는 /Companero-1.2/companero)》를 실행하면 Eclipse를 기동할수 있으나 기동의 편리를 위하여 탁상에 지름아이콘을 만들기로 한다. 《/kut/eclipse/eclipse(또는 /Companero-1.2/companero)》의 지름아이콘을 탁상에 작성하고 지름아이콘이름을 Eclipse3.0 등으로 한다. 이상으로 설치는 완료되였다.

• Eclipse의 기동과 서류철구성

탁상에 작성한 Eclipse의 지름건을 기동하면 작업공간을 지정하는 화면이 나온다. 작업공간등록부는 기정으로 《[Eclipse의 서류철]/workspace》로 되여있으나 관리하기 좋게《/kut/workspace(또는 /Companero-1.2/workspace)》로 지정한다(그림 1-5). 여기서 《Use this as the default and do not ask again(이 선택을 기정으로 사용하고 앞으로는 이 질문이 나타나지 않는다.)》검사칸을 선택하면 다음번부터는 이 대화칸은 나타나지 않게 된다. 만일 기동할 때마다 작업공간을 지정하고싶을 때에는 선택하지 않고 [OK]단추를 찰칵하면 된다.

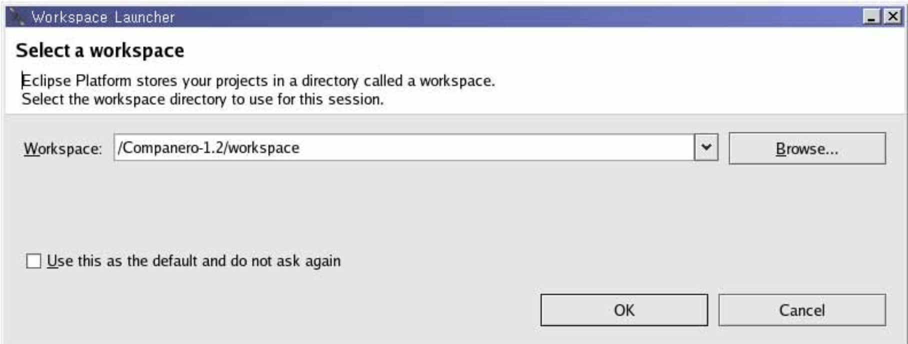


그림 1-5. 작업공간등록부의 선택

Eclipse를 설치한 후 처음으로 기동할 때에는 그림 1-6과 같이 [Welcome]화면이 나타난다.

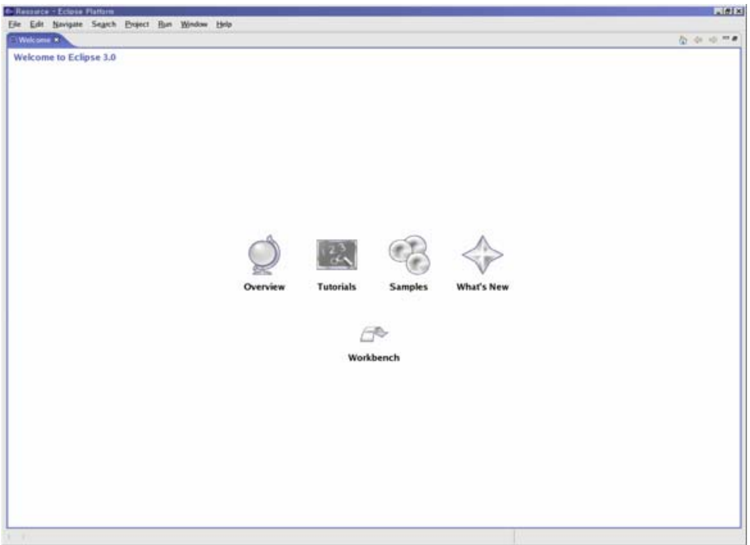


그림 1-6. 첫 기동할 때 Welcome화면

이 화면에서 매개 차림표의 내용은 아래와 같다.

표 1-3. Eclipse의 기동차림표

차림표	내 용
Overview	작업공간의 기본, Java개발, 조지원, Eclipse끼워넣기개발을 위한 도움말에로의 련결(link)을 할수 있다.
Tutorial	Java개발, Java Applet개발, SWT응용프로그람개발, 끼워넣기개발, 특성정보의 갱신거점에로의 배치(deploy)에 관한 개별지도서가 있다.
Samples	SWT의 견본, 작업공간에로의 통합코드견본이 있다.
What's New	Eclipse에 관한 새로운 정보나 이전 판본으로부터의 이행, Eclipse의 갱신을 진행할수 있는 차림표가 있다.
Workbench	Eclipse의 기본작업환경으로 이행할수 있다.

Eclipse를 리용하여 개발을 시작하려면 화면의 오른쪽우의 화살표식(작업공간)을 선택한다. 그러면 그림 1-7과 같은 화면이 나타난다. Eclipse설치후 첫 기동할 때에만 그림 1-6과 같은 Welcome화면이 나타난다. 만일 이 초기화면을 다시 현시하려 한다면 차림표의 [Help]→[Welcome]을 리용하여 실현할수 있다.

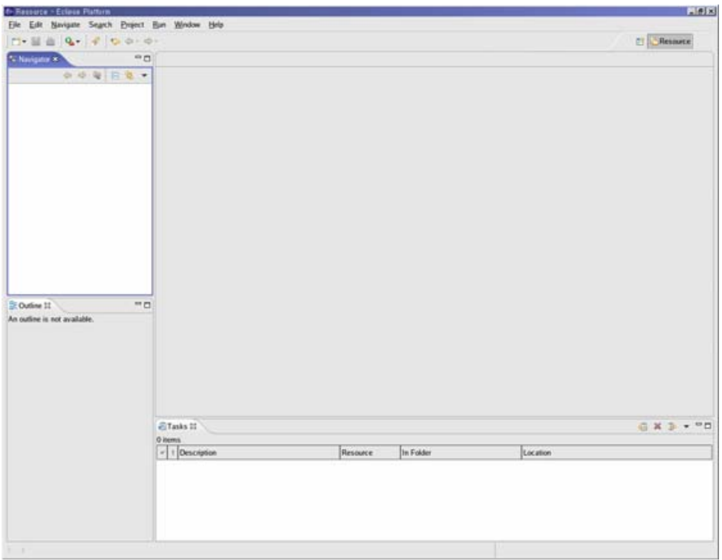


그림 1-7. Eclipse창문

지금까지의 공정으로 《/root》이하의 서류철은 그림 1-8과 같이 되여있다.

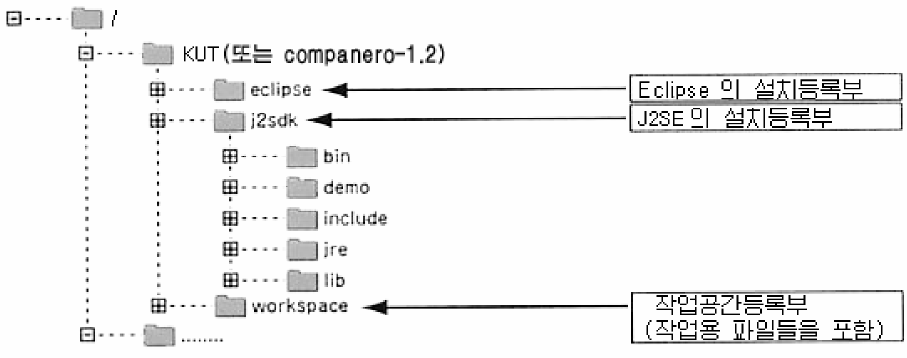


그림 1-8. 설치후 서류철의 내용

제2장. Eclipse가동기반

Eclipse는 끼워넣기를 써서 여러가지 목적을 실현하는 도구이다. 이 장에서는 Eclipse가동기반이 가지고있는 가장 기본적이며 공통적인 기능과 환경에 대하여 설명한다.

제1절. Eclipse의 기본조작

Eclipse를 기동하면 그 창문우에는 여러가지 화면부품이 표시된다. 또한 그 겉모양은 사용자가 쓰고싶어하는 환경으로 변화시킬수 있다. 여기서는 Eclipse의 기본적인 기능과 조작방법에 대하여 취급한다.

2.1.1. 작업대와 투시

Eclipse는 개발자에게 작업대(Workbench)를 제공한다. 이것은 Linux, Windows, Mac OS 등의 조작체계와 같은 관점에서 고찰하면 된다. 조작체계는 그자체로서는 구체적인 작업을 할수 없다. 우리는 조작체계우에 목적에 맞는 응용쏘프트웨어를 설치하고 사용하여야 한다. 마찬가지로 개발자도 개발환경을 작업대우에 전개하여 개발을 진행한다. 바로 이것이 Eclipse상에서 여러가지 개발을 수행할수 있는 리유이다. 기동되면 창문이 나타나는데 이것을 작업대창(Workbench window)이라고 부른다(그림 2-1).

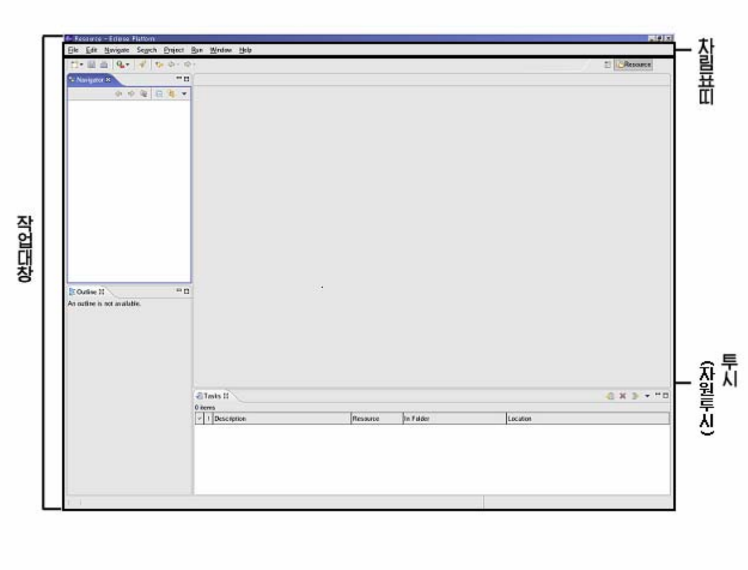


그림 2-1. 작업대와 그 우에 표시된 [Resource]투시

작업대에 나타나는 환경은 《투시(Perspective)》라고 부르는 단위로 관리한다. 개발을 진행할 때에는 여러개의 투시를 열수 있다. 설치직후에는 [Resource]라는 투시가 표시된다. 이외에도 표준장비되여있는 투시(표 2-1참고)가 있기때문에 개발자는 목적에 맞게 투시를 절환하여 보다 효률적으로 작업을 진행할수 있다. 또한 끼워넣기도구([2.4 끼워넣기의 설치]참조)에 의하여 투시를 추가할수도 있다. 외부끼워넣기를 사용함으로써 종래의 개발환경과는 달리 Eclipse에서는 단일한 쏘프트웨어(Eclipse)상에 여러개의 개발환경을 갖추는것도 가능하게 되였다. 이렇게 함으로써 개발환경이 달라질 때마다 도구의 사용법을 다시 배우는것과 같은 복잡한 공정을 없앨수 있다.

표 2-1. 표준투시일람

투 시 명	설 명
CVS Repository Exploring	CVS의 보관고관리를 진행한다.
Java	Java개발을 진행한다.
Java Browsing	Java의 클라스 및 패키지구성을 확인하면서 Java의 개발과 관리를 진행한다.
Java Type Hierarchy	Java형의 계층검색을 진행한다.
Team Synchronizing	팀개발에서 원천동기를 맞추는 작업을 진행한다
Debug	Java의 국부오유수정, 원격오유수정를 진행한다.
Plug-in Development	끼워넣기개발을 진행한다.
Resource	원천파일 등 자원관리를 진행한다.

이제부터 실지로 현재의 투시(설치직후이면 자원투시)로부터 다른 투시(여기서는 Java투시)로 절환해보기로 한다.

차림표띠에서 [Window]→[Open Perspective]→[Other]를 선택하여 투시일람표를 현시한다. 여기에는 표준으로 설정되여있는 투시일람이 있다(그림 2-2). 그 일람에서 [Java]를 선택하고 [OK]를 찰칵한다. 작업대창의 현시내용이 갱신되고 Java투시가 작업대우에 전개된다(그림 2-3). 다시 [Resource]투시로 되돌리려면 오른쪽우에 있는 투시절환도구단추를 찰칵한다.

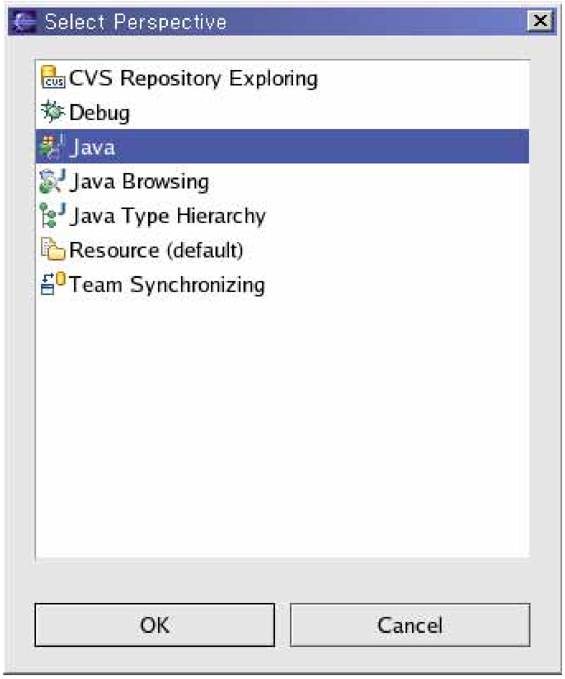


그림 2-2. 투시의 선택

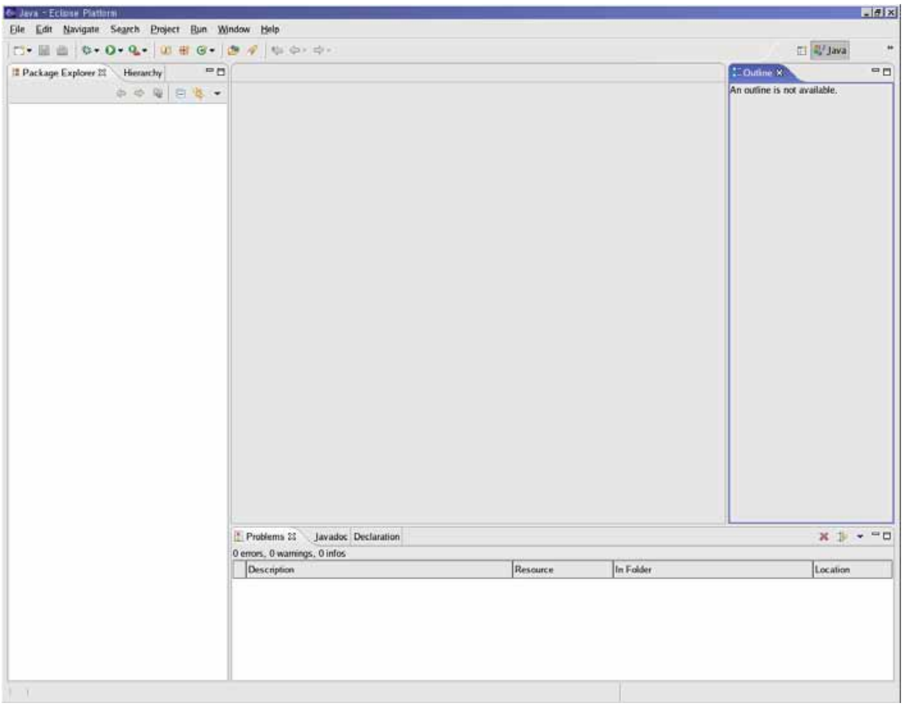


그림 2-3. Java투시

2.1.2. 보기(view)와 편집기

투시는 여러개의 보기와 편집기로 구성되여있다(그림 2-4). **보기**란 Windows의 열람기처럼 파일이나 패키지의 계층구조, 오유수정중의 정보, 능동상태에서 편집기의 속성(property)을 표시한것을 말한다. 초기화면의 자원투시에 표시되는 [navigator]나 [task] 등이 여기에 해당된다. 표 2-2는 많이 쓰이는 대표적인 보기의 일람이다. 보기의 종류는 표 2-2외에도 수많이 있으나 그것을 다 외워둘 필요는 없다. 그 리유는 매개 투시가 최소한 필요하다고 보는 보기를 표준장비하고있기때문이다. 만일 부족되는 보기가 있을 때에는 개별적으로 투시에 추가할수 있다.

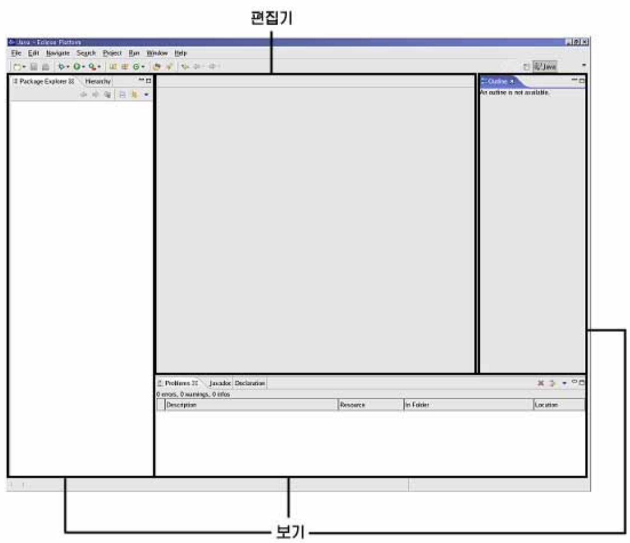


그림 2-4. 보기와 편집기

표 2-2. 보기일람

보기명	설명
Ant	build.xml의 속성을 현시하며 Ant를 실행한다.
Bookmark	작성한 책갈피를 현시한다.
Navigator	작업대안의 자원의 계층구조를 현시한다.
Outline	클라스의 마당(field)이나 메쏘드(method)를 현시한다.
Property	편집기에 열려져있는 자원의 속성이름과 값 등을 현시한다.
Synchronizing	CVS보관고와 국부자원의 변경정보나 동기처리내용을 표시한다.
Task	콤파일오유나 TODO목록을 나타낸다.

편집기에서는 자원의 현시와 편집을 진행한다. Java파일이나 Ant의 build.xml파일에 대하여서는 표준으로 편집기가 붙어있어서 예약어를 강조문자로 표시하며 오유가 있는 경우 빨간 파도선으로 오유위치를 표시하여준다.

매 투시마다 보기와 편집기의 구성이 설정되여있는데 새로 보기를 추가하거나 현재 설정되여있는 보기를 현시하지 않도록 하는것과같은 투시의 전용화를 진행할수 있다. 례를 들어 자원투시안의 [Ant]보기를 현시하려면 차림표띠에서 [Window]→[Show View]→[Ant]를 선택한다(그림 2-5).

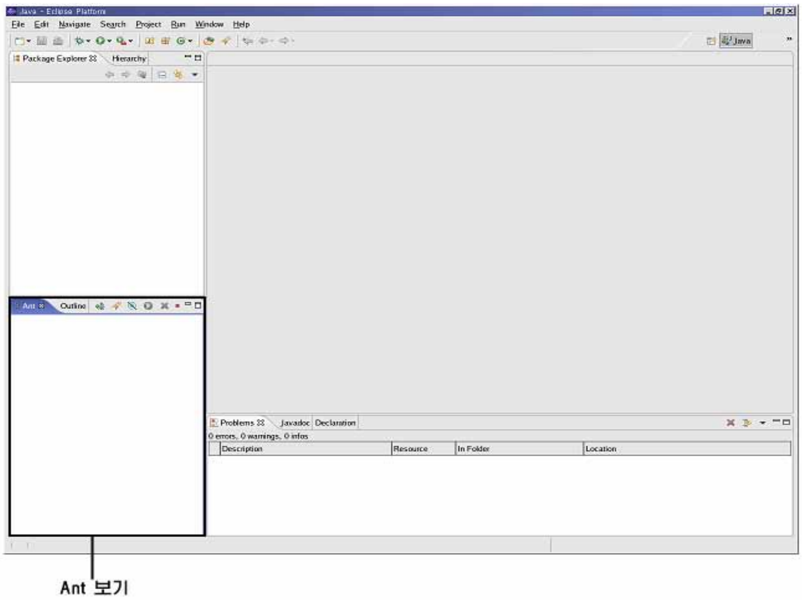


그림 2-5. Ant보기의 현시

또한 보기의 현시방법으로서 필요할 때에만 현시하게 하는 **고속보기**가 있다. 이것은 보통 보기와는 달리 단추화된 비표시보기이므로 작업대의 공간을 차지하지 않는다. 이것은 보기를 많이 리용하면서도 작업령역을 확보하려고할 때 효과적인 방법이다.

고속보기의 작성방법은 고속보기로 하려고하는 보기의 제목띠에서 마우스오른쪽단추를 찰칵하면 나타나는 차림표(이 차림표를 지름차림표라고 한다.)의 [Fast View]를 선택한다. 그러면 창문아래쪽에 고속보기단추가 나타나고 보기가 비표시로 된다. 이 단추를 찰칵하면 해당보기가 일시적으로 표시되게 된다.

2.1.3. 도구띠와 단추

투시나 그 안에 있는 보기, 편집기의 기능을 리용하거나 최량화하기 위하여 도구띠와 단추를 리용한다. Eclipse에는 아래와 같은 도구띠와 단추가 있다(그림 2-6).

- 주요도구띠(main tool bar)

이것은 작업대창의 웃쪽에 있는 차림표띠의 바로 아래에 표시되는 도구띠이다. 새로운 프로젝트의 작성이나 보관과 같은 Eclipse의 기본기능과 프로그람의 실행 등을 할수 있다. 이 도구띠는 투시단위로 설정된다. 또한 단추의 구분부분을 끌어놓기하여 배치를 바꿀수도 있다.

- 보기도구띠(view tool bar)

매개 보기의 기능을 리용하기 위한 도구띠이다. 매개 보기의 제목띠에 표시되여있다. 아이콘이나 [▼]으로 표시되는 차림표로부터 보기의 현시설정을 변경할수 있다.

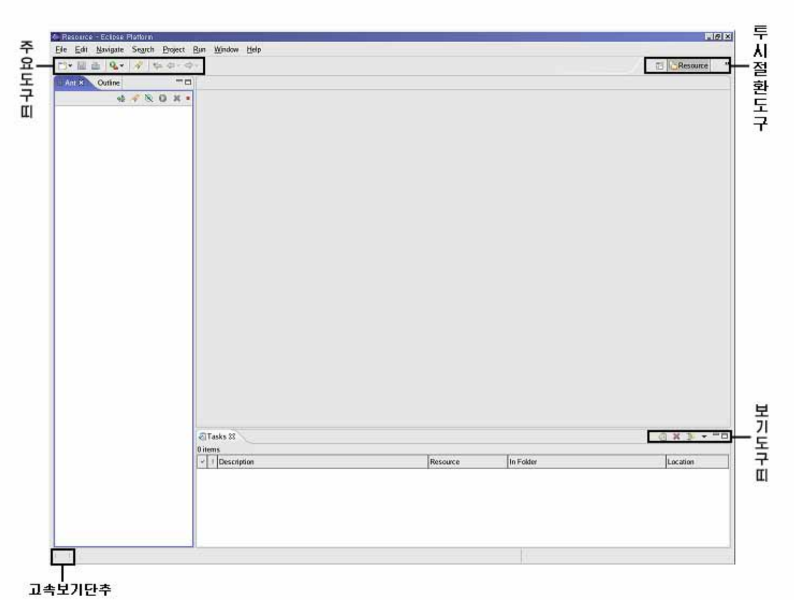


그림 2-6. 도구띠

- 투시절환도구

투시절환도구는 현재 열려있는 투시를 간단히 절환하기 위한 도구띠이다. 작업대창의 오른쪽우에 표시된다. 투시를 열 때마다 단추의 아이콘이 추가되고 투시를 닫으면 없어진다.

- 고속보기단추

고속보기를 실현할수 있는 단추이다. 고속보기단추는 창문의 왼쪽, 아래, 오른쪽중의 어느 하나로 이동될수 있다.

2.1.4. 화면의 전용화

제한된 화면치수나 해상도의 조건에서 개발을 하려고할 때 개발자자신이 요구하는 화면환경으로 바꿀수 있다. Eclipse에서는 투시안의 보기종류나 현시위치, 크기를 변경하거나 단추, 차림표를 변경하는 등 개발자에게 있어서 가장 적합한 환경을 구비할수 있도록 투시를 전용화할수 있다.

- 보기의 재배치

재배치는 마우스로 보기의 제목띠를 끌어놓기하면 된다. 보기의 끌기를 시작하면 놓기유표가 표시된다. 놓기유표가 표시되였을 때 단추를 놓으면 그 형태에 따라 표 2-3에서 서술한 동작이 일어난다. 또한 보기의 크기변경도 가능한데 그 경우 린접한 보기들의 크기도 동적으로 변화된다는데 주의해야 한다. Eclipse3.0부터는 창문밖으로 끌어놓기함으로써 편집기밖에 독립적으로 보기창문을 현시하는것도 가능하다.

표 2-3. 놓기유표일람

놓기유표	설 명
⬆(우로 이동)	보기의 제목띠를 선택한 상태에서 우로 끌면 이 유표가 현시된다. 이때 눌렀던 단추를 놓으면 보기창은 원래 자리의 웃쪽에 배치된다.
⬇(아래로 이동)	보기의 제목띠를 선택한 상태에서 아래로 끌면 이 유표가 현시된다. 이때 눌렀던 단추를 놓으면 보기창은 원래 자리의 아래쪽에 배치된다.
➡(오른쪽으로 이동)	보기의 제목띠를 선택한 상태에서 오른쪽으로 끌면 이 유표가 현시된다. 이때 눌렀던 단추를 놓으면 보기창은 원래 자리의 오른쪽에 배치된다.
⬅(왼쪽으로 이동)	보기의 제목띠를 선택한 상태에서 왼쪽으로 끌면 이 유표가 현시된다. 이때 눌렀던 단추를 놓으면 보기창은 원래 자리의 왼쪽에 배치된다.
(겹쳐서 현시)	배치하려는 자리에 있는 보기창의 내부로 끌기하면 이 유표가 현시된다. 이때 눌렀던 단추를 놓으면 현재 위치에 있던 보기창의 안쪽에 이동되여 온 보기창이 태브표시로서 배치된다.
(보기의 독립화)	배치하려는 보기를 창문으로부터 독립적으로 표시한다.
⃠ (떨구기금지)	편집기상이나 배치불가능한 자리에서 표시되며 이 표시중에 단추를 놓아도 보기는 이동하지 않는다.

보기의 오른쪽우에 있는 최소화단추를 찰칵하면 보기를 접어서 표시한다. 또한 최대화단추를 찰칵하면 보기가 투시전체에 표시된다(그림 2-7).

편집기도 보기와 마찬가지로 끌어놓기로 편집기령역을 여러개로 분할할수 있다. 그러나 편집기전체의 령역은 보기의 배치에 의존하여 현시위치나 령역의 넓이가 변하기때문에 보기의 재배치를 진행할 때에는 편집기의 령역을 고려할 필요가 있다.

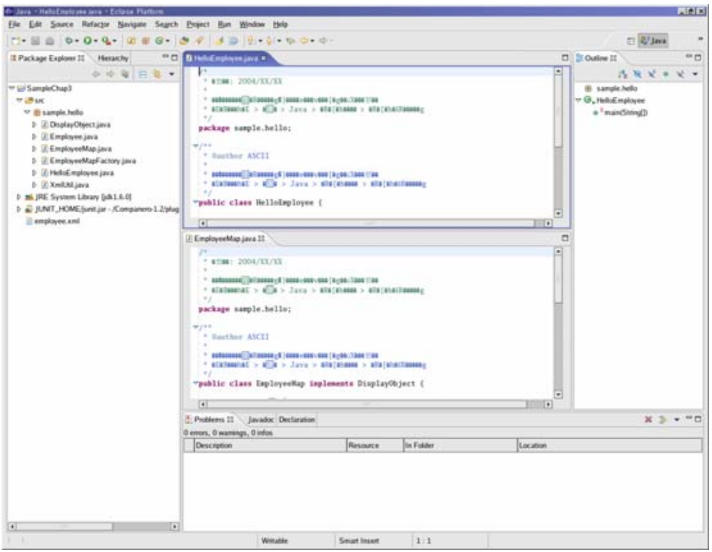


그림 2-7. 보기의 최소화, 편집기의 재배치

- 투시의 전용화

현재의 투시에 사용자에게 편리한 새로운 기능을 추가할수 있다. 차림표띠의 [Window]→[Customize Perspective]를 선택하면 그림 2-8이 나타난다.

Shortcuts태브에서는 부분차림표나 지름건차림표에 표시하는 항목을 전용화할수 있다. Commands태브에서는 차림표띠의 항목이나 도구띠단추의 유무를 전용화할수 있다. 이러한 설정내용은 투시별로 보존된다.

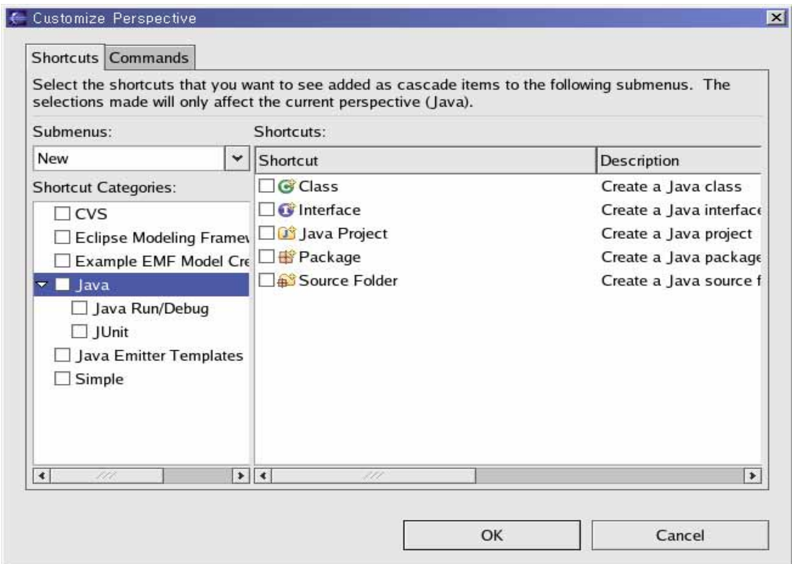


그림 2-8. 투시의 전용화

보기의 려과기기능

[Package Explorer]보기나 [Navigator]보기에는 프로젝트정보가 있다. 특히 Java개발에서 사용하는 [Package Explorer]보기에는 사용하는 서고파일(*.jar)이나 Java의 원천파일뿐만아니라 다른 파일들도 현시되기때문에 서고파일을 많이 사용할 때 보기 힘들어지는 경우가 있다. 이러한 경우에 보기의 차림표에 있는 [Filter]를 리용하는것이 좋다. 이 기능에 의하여 지정한 조건을 만족하는 파일만을 보기에 현시할수 있다. 그림 2-9는 [Package Explorer]보기의 려과기설정화면이다. 검사칸을 선택함으로써 원하는 파일만을 표시할수 있다.

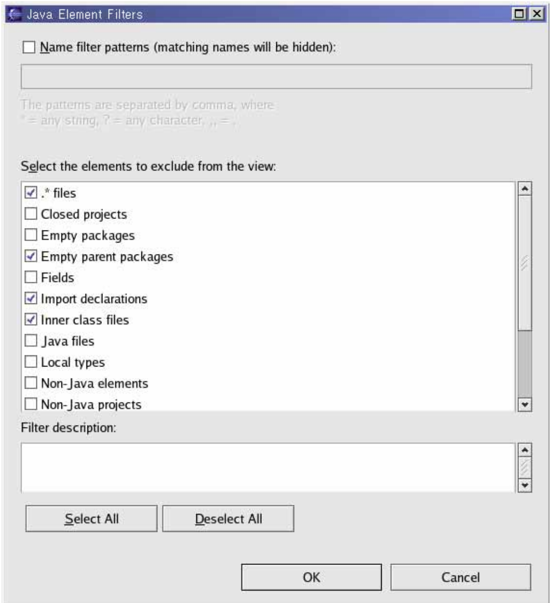


그림 2-9. Java요소려과기

제2절. 환경설정

편집기의 서체나 건반의 설정 등 Eclipse의 환경설정은 차림표띠의 [Window]→[Preferences]을 선택하여 현시되는 대화칸에서 진행할수 있다. 이 대화칸에서는 작업대의 설정, Java나 Ant, 도움말 등의 설정도 할수 있다. 여기서는 작업대와 Java개발을 위한 주요한 설정에 대하여 설명한다.

2.2.1. JRE의 정의 추가와 기정JRE의 설정

Eclipse에서는 Java프로그람의 작성이나 실행에 사용하는 Java실행환경을 여러개 등록해놓고 프로젝트의 속성이나 기동구성에서 지정함으로써 각이한 Java실행환경을 사용할수 있다. 여기서는 제1장에서 설명한 SDK에 포함되는 JRE를 기정 JRE로 설정하는 방법에 대하여 설명한다.

우선 JRE의 정의를 Eclipse에 추가한다. 설정순서는 다음과 같다.

1. 차림표띠의 [Window]→[Preferences]을 선택한다.
2. 대화칸의 왼쪽에서 [Java]→[Installed JREs]를 선택한다(그림 2-10).

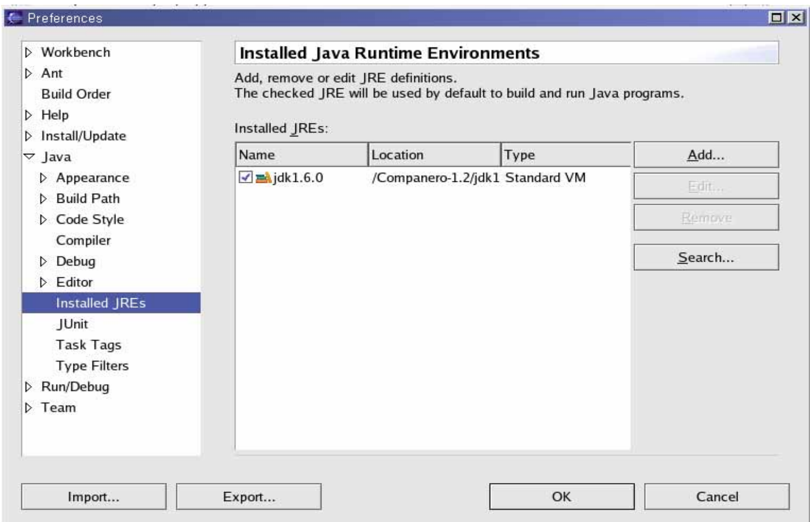


그림 2-10. JRE의 설정대화칸

3. 대화칸의 오른쪽에서 [Add]단추를 찰칵한다.
4. 이때 현시되는 대화칸의 [JRE type]에서 《Standard VM》을 선택하고 [JRE name]에 《jdk1.6.0》, [JRE home directory]에 《/Companero-1.2/jdk1.6.0》(SDK를 설치한 경로)를 입력한다. 또한 [Use default sytem libraries]검사칸을 선택한다.

그러면 [JRE home directory:]에 입력된 경로에 JRE가 설치되여있는가를 검사하고 자동적으로 Javadoc URL의 설정과 JRE체계서고의 jar파일의 설정이 지정된다(그림 2-11).

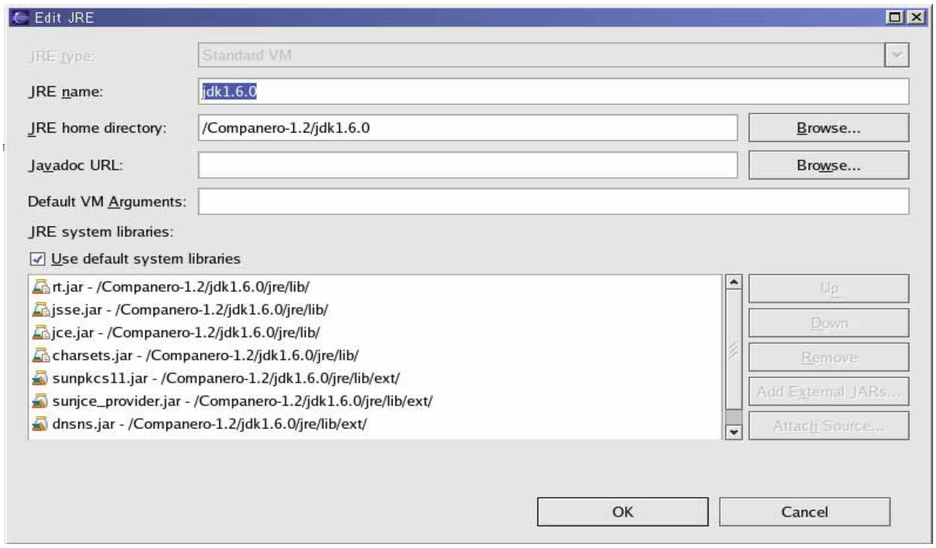


그림 2-11. JRE의 작성대화칸

5. [OK]단추를 찰칵하여 대화칸을 닫으면 이미 설치되여있는 JRE의 일람에 새로운 JRE의 정의가 추가된다.

이상으로 JRE의 정의가 작성되였고 계속하여 기정JRE에 대한 변경을 진행한다.

1. 대화칸의 [Installed JREs]에서 앞서 등록한 [jdk1.6.0]검사칸을 선택한다.
2. 대화칸의 [OK]단추를 찰칵한다.
3. 변경을 유효로 하기 위한 확인대화칸이 표시되는데 [Yes]단추를 찰칵한다.

개발대상의 JDK판본이 하나일 때에는 JRE의 변경은 특별히 필요없다. 그러나 콤퓨터의 환경에 따라서는 JRE만 있는 서류철에 기정JRE가 지정되는 경우도 있다. 이러한 상태에서도 개발을 진행할수는 있으나 SDK에 포함되는 원천코드가 추가되지 않는다. 우에서 보는바와 같이 SDK에 포함된 JRE로 변경하면 SDK와 함께 배포되여있는 원천코드가 추가되기때문에 JRE체계서고의 원천코드를 참조할수 있게 된다.

JRE의 원천코드는 Java개발자들이 직접 쓴것들이기때문에 코드분석을 위한 리상적인 교재라고 할수 있다. SDK를 설치하고있다면 여기서 설명한 절차를 따라 기정JRE를 변경하는것이 좋다.

2.2.2. 기동할 때 추가선택에 의한 환경설정

Eclipse를 기동할 때에 추가선택을 진행함으로써 여러가지 설정을 변경할수 있다. 례를 들어 작업공간등록부는 보통 Eclipse를 기동할 때에 작업공간설정대화칸에서 설정한다(그림 2-12).

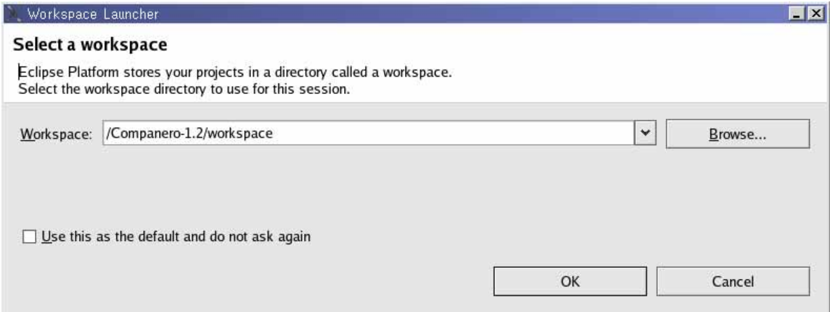


그림 2-12. 작업공간의 선택

기동할 때 작업공간의 선택은 한번 지정하면 기억된다. 설정정보는 《/kut/eclipse/configuration/org.eclipse.ui.ide/recent/Workspaces.xml》에 기록된다. 기동할 때의 작업공간설정대화칸에서 [Use this as the default and do not ask again] 검사칸을 선택하면 작업공간설정대화칸이 표시되지 않으며 recentWorkspaces.xml파일안의 《alwaysAsk showDialog="0"/》에서 《0》을 《1》로 설정하여 다시 작업공간설정대화칸을 현시할수 있다.

작업공간등록부의 지정은 Eclipse의 실행파일인수에 《-data》를 지정하는 방법으로도 가능하다.

이 책에서 취급하는 작업공간등록부로는 우에서 설정한 《/kut/workspace》를 사용하고 제3장이후에 작성하는 견본코드(Java파일, xml파일 등)는 모두 《/kut/workspace》아래에 작성할것을 전제로 한다.

/kut/eclipse/eclipse -data <설정하는 작업공간등록부의 경로>

Eclipse의 실행에 쓰이는 가상기계는 환경변수의 Path로부터 자동적으로 검색되지만 다른 가상기계가 지정되여있어서 제대로 기동되지 않았거나 여러가지 가상기계를 엇바꾸어 쓰고싶을 때는 다음과 같은 방법으로 가상기계를 지정할수 있다.

/kut/eclipse/eclipse -vm <VM의 실행파일>

또한 Eclipse의 실행에 쓰이는 VM의 인수를 리용하고싶을 때는 《-vmargs》를 지정한다. 대규모응용프로그람이나 여러개의 프로젝트를 동시에 개발하려고할 때는 기억기크기를 늘여서 실행하는것이 좋다.

/kut/eclipse/eclipse -vmargs <VM의 인수를 서술>

Eclipse의 실행시간고속완충자료를 지우기 위해서는 《-clean》을 지정한다. 새로운 끼워넣기를 넣었을 때나 작업공간의 파일을 강제적으로 복사했을 때 비정상적인 현상이 일어난 경우 이것을 리용하면 개선될수 있다. 《-clean》을 지정하면 실행시간작업대의 모

든 실행시간고속완충자료가 기동할 때마다 지워지고 작업대에서 진행된 모든 변경이 실행시간작업대가 기동할 때에 반영되게 된다. 그대신 끼워넣기를 많이 포함하는 경우에 기동할 때의 성능이 낮아질수 있다.

/kut/eclipse/eclipse -clean

다른 인수에 대한 설명은 도움말의 [Workbench User Guid]→[Task]→[Running Eclipse]에 있다.

또한 clean인수에 대하여서는 도움말의 [PDE Guide]→[Reference]→ [Tips and tricks]에 있다.

- Eclipse를 기동시켰을 때 장애해결방법

Eclipse가 제대로 기동하지 않는 가장 많은 경우가 여러개의 가상기계가 설치되여있는 환경에서 Eclipse의 기동에 쓰이는 가상기계가 대응되여있지 않은 경우이다. 례하면 JRE1.3.1과 JRE1.4.2가 설치되여있는 환경에서 Eclipse의 기동에 JRE1.3.1을 리용하면 《An Error has occurred. ……(오유가 발생하였습니다. ……)》 등의 통보문이 현시되면서 기동하지 않는다. 이러한 경우에는 기동추가선택을 써서 명시적으로 리용할 가상기계를 지정하면 해결된다. 구체적으로는 기동추가선택에 《 -vm /usr/java/jdk/jre/bin/java》로 지정하여주면 된다.

2.2.3. 작업대의 환경설정

작업대나 매개 투시의 환경설정은 차림표의 [Window]→[Preferences]을 선택하면 그림 2-13과 같은 설정대화칸이 표시되는데 여기에서 진행한다. 매 항목의 왼쪽에 있는 ▷단추를 찰칵하면 상세한 설정항목이 전개된다. 표 2-4에서 Preferences대화칸의 항목과 설정내용을 보여주었다. 설정파일은 [import], [export]단추를 리용하여 읽어들이고 보존할수 있다.

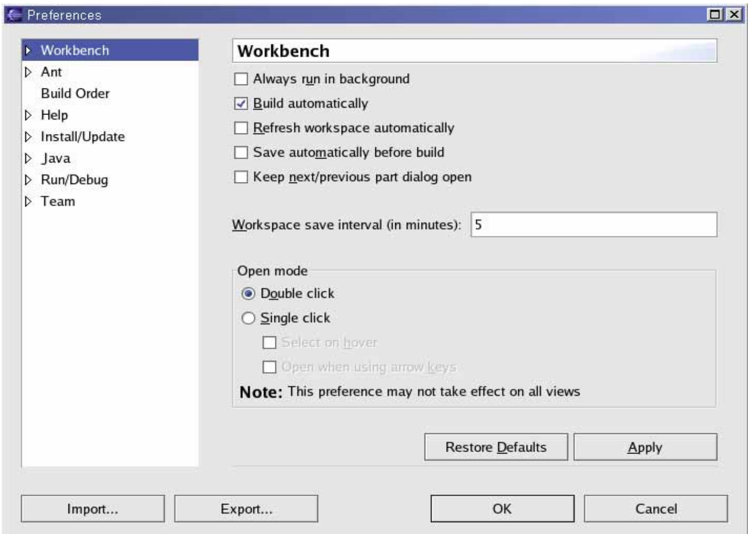


그림 2-13. 설정대화칸

표 2-4. 설정항목과 설정내용

항목명	설정내용
Workbench	본문편집기, 투시, 서체, 건(key) 등의 설정
Ant	Ant편집기(build.xml), 실행시 클라스경로의 설정
Java	Java편집기, 코드형식화기, JUnit 등의 설정
Install/Update	갱신할 때 대리(proxy)의 설정, 자동갱신의 설정 등
Team	CVS, 파일내용물 등의 조기능설정
Build Order	프로젝트건립순서의 설정
Help	도움말열람기의 설정, 도움말봉사기의 포구설정 등
Run/Debug	조종탁(console), 보기관리, 기동 등 실행 및 오유수정기능의 설정

여기서는 작업대에 관한 기본설정에 대하여 소개한다.

- Workbench(작업대)

작업대에 대한 기본적인 설정을 진행할수 있다(그림 2-14). 표준설정에서는 원천파일을 보존하면 동시에 콤파일이 진행되지만 [Build automatically]검사칸을 선택하지 않으면 수동으로 콤파일을 진행해야 한다.

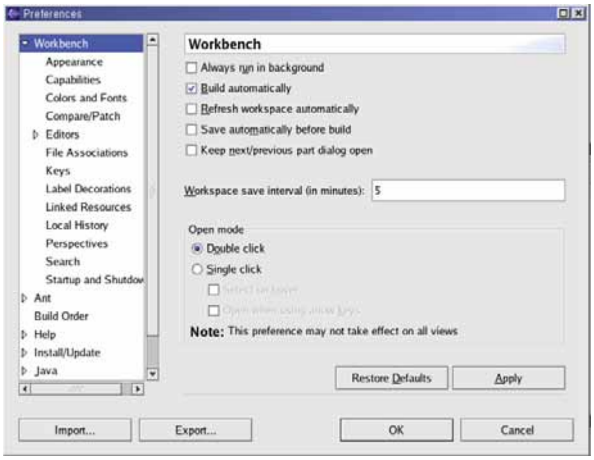


그림 2-14. 작업대의 설정

- Editors(편집기)

편집기의 부호화지정이나 편집기안의 행번호표시, 편집기안의 문자표시색의 설정 등을 할수 있다(그림 2-15). 본문편집기의 항목에 있는 [Show line numbers]검사칸을 선택하면 편집기의 왼쪽에 행번호가 표시된다.

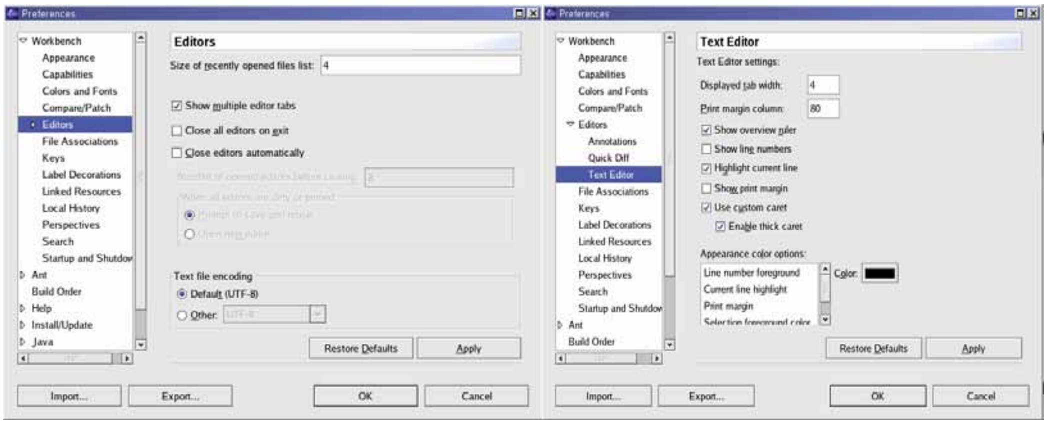


그림 2-15. 본문편집기의 설정

• Appearance(겉모양)

편집기의 태브위치에 대한 설정을 할수 있다(그림 2-16). 례하면 편집기태브는 표준으로는 웃쪽에 표시되지만 [Editor tab positions]나 [View tab positions]의 선택표시를 Bottom(아래쪽)으로 바꾸면 태브의 위치가 아래쪽으로 이동한다.

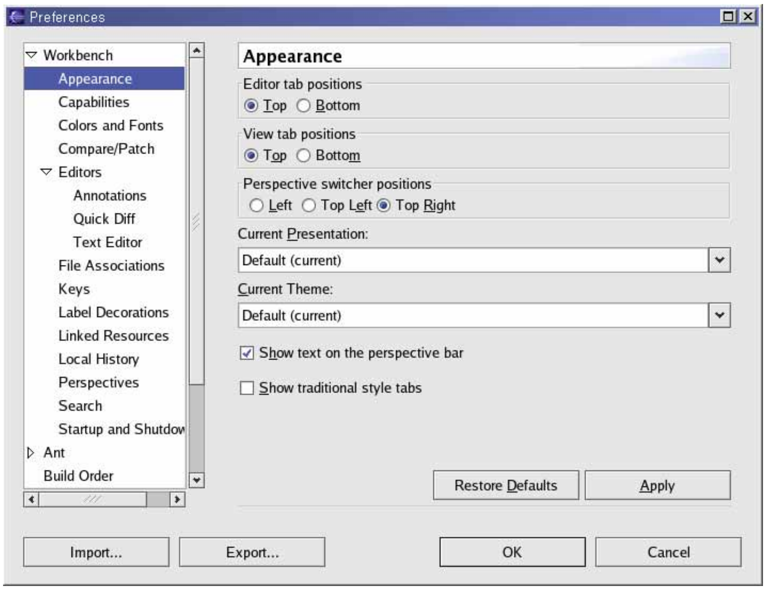


그림 2-16. 겉모양의 설정

• Colors and Fonts(색과 서체)

대화칸이나 보기, 편집기의 색과 서체를 지정할수 있다(그림 2-17). [Colors and

Fonts]에 검색문자를 입력하거나 아래의 목록에서 대상으로 되는 설정항목을 골라서 [Change]단추를 찰각하면 설정대화칸이 표시된다. 대화칸에서 필요한 서체를 선택할수 있다.

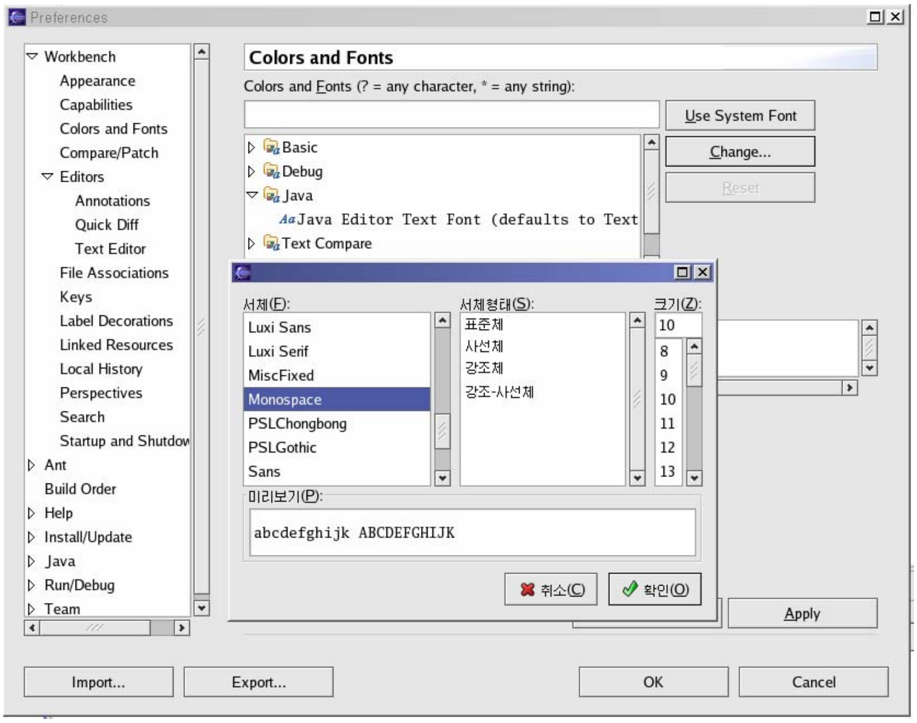


그림 2-17. 색과 서체설정

• Keys(건)

건반에 관한 설정을 할수 있다. 표준으로는《Default》와《Emacs》으로 되여있다.

또한 건에 대한 전용설정을 하자면 그림 2-18과 같이 하면 된다. [Command]에서 추가 또는 변경하려는 범주(category), 이름, 할당을 선택하여 아래단의 [key sequence]에서 건의 할당을 추가 또는 삭제한다. [Add]단추를 찰각하면 웃단의 [Assignments]에 추가된다. [Apply]를 찰각하면 설정이 반영된다.

개발자에게는 건반작업에서 지름건은 필수적이기때문에 자기의 작업상 편리를 고려하여 전용화하는것이 좋다.

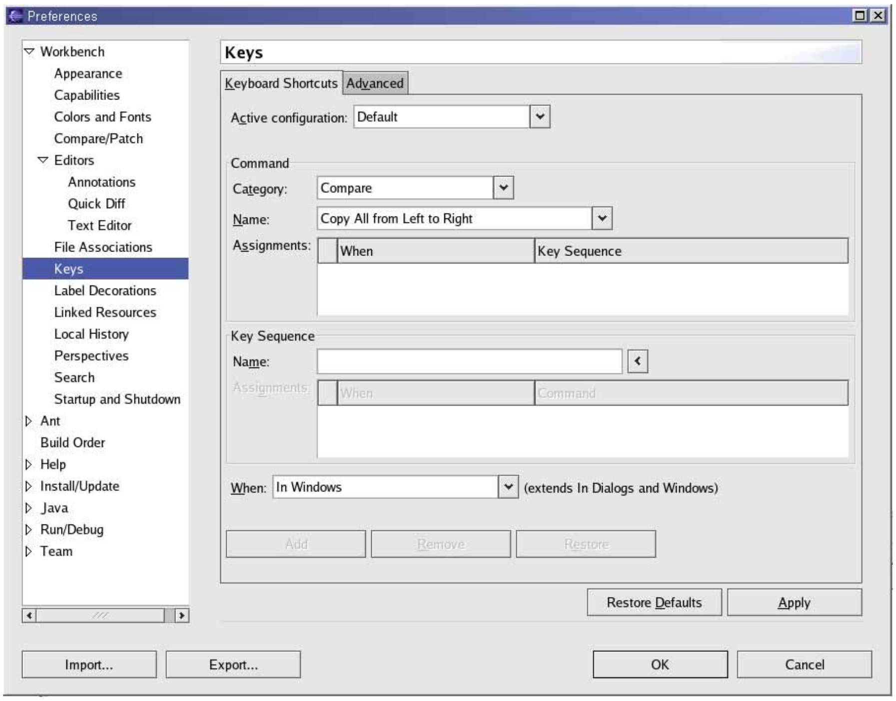


그림 2-18. 건설정

2가지 설정

Eclipse에서는 2가지 중요한 설정을 진행하여야 한다.

하나는 작업대에 관한 설정이다. 이것은 JVM나 끼워넣기도구, 작업대의 전반적인 설정이고 작업공간단위로 관리된다.

다른 하나는 프로젝트에 관한 설정으로서 프로젝트의 속성에서 진행할수 있다. 여기서는 프로젝트단위로 리용하는 클라스경로나 콤파일 설정을 진행할수 있으며 그 내용은 매개 프로젝트등록부의 바로아래에 있는 .project, .classpath에 서술되여있다.

Java콤파일러나 과제 등의 설정은 작업대나 프로젝트에서 진행할수 있는데 량쪽 다 설정되여있는 경우에는 프로젝트의 설정이 우선권을 가진다. 프로젝트에서 설정하지 않는 경우에는 작업대에서의 설정이 적용된다.

제3절. 도움말의 사용법

Eclipse는 도움말기능이 풍부하여 Eclipse의 기본적인 안내서로부터 Eclipse의 끼워넣기개발에 관한 안내서에 이르기까지 폭넓은 정보를 제공한다. 또한 검색이나 책갈피와 같은 웨브열람기의 기능도 있다. 여기서는 도움말의 내용과 그 기능에 대하여 설명한다.

2.3.1. 도움말열람기

도움말열람기를 보기 위해서는 차림표띠에서 [Help]→[Help Contents]를 선택한다. 그러면 Eclipse와는 다른 창문에 도움말열람기가 현시된다(그림 2-19). 처음에 현시되는 보기를 **도움말보기**(Help view)라고 부른다. 도움말보기에서는 차례나 검색기능을 리용하여 정보를 검색할수 있다.

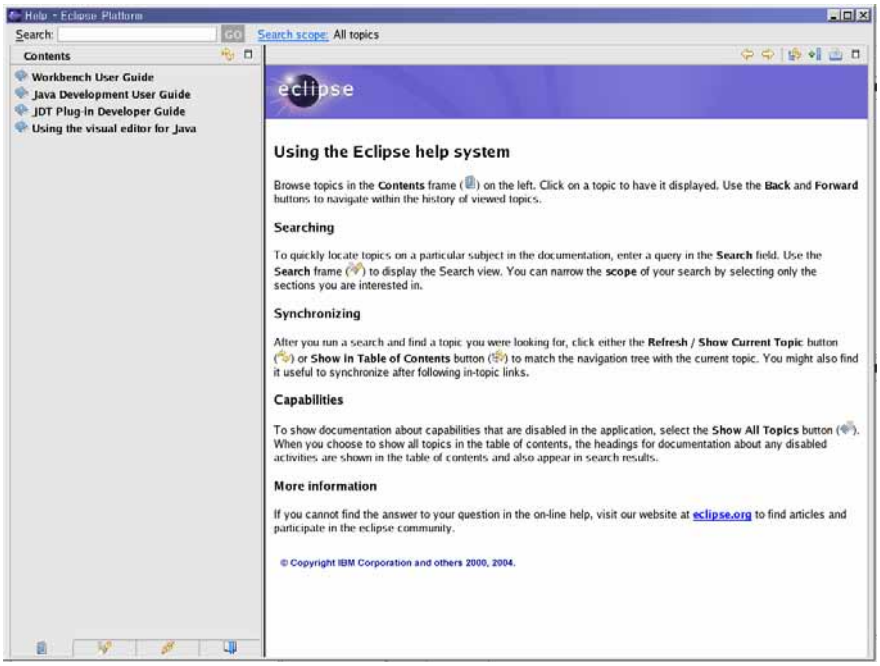


그림 2-19. 도움말열람기

문맥(context)에 의존한 도움말의 리용

Microsoft Office 등의 일반적인 Windows응용제품들과 마찬가지로 F1건을 리용함으로써 필요한 정보를 재빨리 검색할수 있다. 이 경우 능동상태로 되여있는 기능에 대응한 도움말이 현시된다. 례하면 [Package Explorer]보기에 관한 정보를 알고싶을 때는 [Package Explorer]보기를 활성화(제목띠를 한번 찰칵)하고 F1건을 누르면 풍선도움말(balloon)이 현시된다. 거기에는 [Package Explorer]보기에 관한 도움말정보가 표시되여있기때문에 알고싶은 항목을 선택하면 된다. 이와 같은 도움말기능을 문맥에 의존한 도움말이라고 한다.

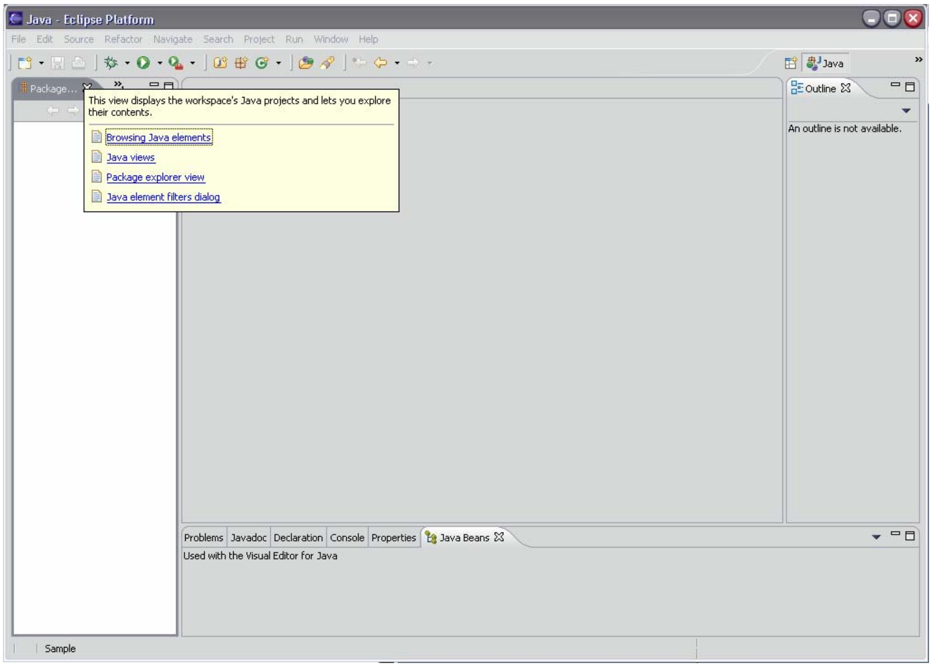


그림 2-20 문맥에 의존한 도움말

2.3.2. 내용물안내

도움말열람기를 기동하면 왼쪽에는 내용물의 일람이 표시된다. 내용물에는 아래와 같은것이 있다.

- 작업대사용자지도서

프로젝트의 작성절차, 편집기나 보기의 사용법 등 Eclipse의 기본적인 설명과 조작법이 서술되여있다. 개별지도서형식으로 되여있으므로 처음으로 Eclipse를 쓰는 사람이라도 알기 쉬운 내용으로 되여있다. 쉽게 기본적인 조작을 습득하는데 리용할수 있다.

• Java개발사용자지도서

프로젝트작성으로부터 실행 및 오유수정까지의 일련의 Java개발에서 리용하는 Eclipse의 Java개발도구(JDT)에 관한 정보가 서술되여있다.

• 가동기반끼워넣기개발자안내서

Eclipse가동기반에 관한 보다 핵심적인 정보를 제공한다. 례를 들어 Eclipse의 개념과 내용에 대한 정보를 제공한다.

• JDT끼워넣기개발자안내서

JDT(Java개발도구)의 끼워넣기도구를 작성하기 위한 구체적인 정보를 제공한다. Eclipse에서 사용하고있는 JDT의 내용을 API의 소개와 함께 해설하고있다.

• PDE안내서

끼워넣기개발환경(Plug-in Development Environment)을 써서 끼워넣기([2.4끼워넣기의 설치]를 참고)를 개발 및 작성할 때 필요한 정보를 제공한다.

2.3.3. 도움말의 검색방법

특정한 어휘나 항목에 대한 정보를 검색하려고할 때에는 도움말열람기의 검색기능을 리용한다. 열람기의 웃쪽에 있는 [Search]부분에 검색하려는 단어를 입력하고 [Search]단추를 찰각하면 해당한 정보가 적중률(hit ratio)과 함께 왼쪽의 도움말보기에 표시된다(그림 2-21). 첫번째 검색때는 색인을 작성하기때문에 좀 시간이 걸리고 다음번부터는 더 빨리 검색할수 있다.

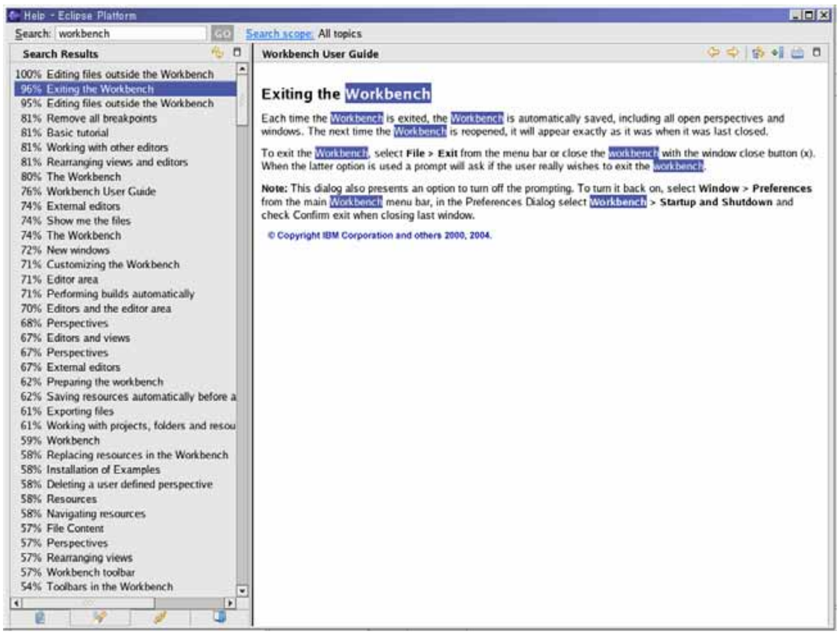


그림 2-21. 도움말의 검색

2.3.4. 조사한 정보를 책갈피에 추가

한번 조사한 정보를 책갈피에 등록함으로써 자주 쓰이는 도움말정보로 설정해놓을수 있다. 열람기의 오른쪽우에 있는 [Bookmark Document]를 누르면 지금 현시되여있는 정보가 책갈피에 등록된다. 그다음부터는 도움말보기의 [Bookmark]단추로부터 정보를 얻을수 있다(그림 2-22).

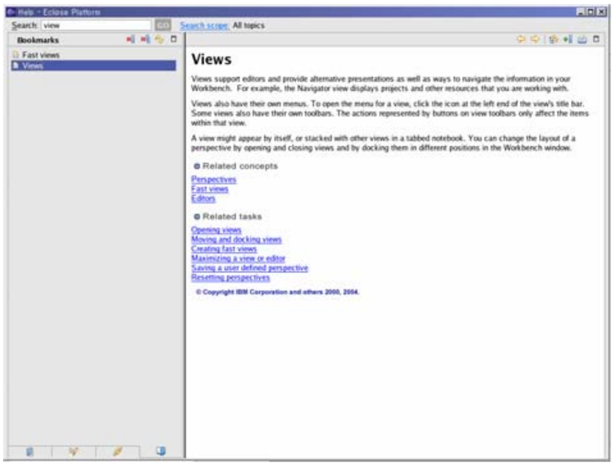


그림 2-22. 한번 조사한 정보를 책갈피에 추가

제4절. 끼워넣기의 설치

2.4.1. 끼워넣기란

끼워넣기란 Eclipse의 기능을 확장하기 위한 모듈이며 Eclipse의 기능은 모두 끼워넣기에 의하여 실현되고있다. 례하면 표준적인 Java개발환경(JDT)이나 팀개발기능들도 다 끼워넣기로 구성되여있다. 또한 외부의 끼워넣기를 설치함으로써 C++의 개발환경을 추가하거나 UML를 작성할수 있다. 끼워넣기의 개발자체도 Eclipse상에서 진행할수 있다. 끼워넣기개발에 대해서는 아래장에 구체적으로 서술되여있으며 여기서는 인터네트등에서 공개되여있는 끼워넣기의 설치방법과 Eclipse의 판본갱신방법에 대하여 설명한다.

- 끼워넣기구성의 관리

차림표의 [Help]→[Software Updates]→[Manage Configration]를 선택하면 이미 설치되여있는 끼워넣기의 구성관리를 진행하는 창문이 나타난다(그림 2-23).

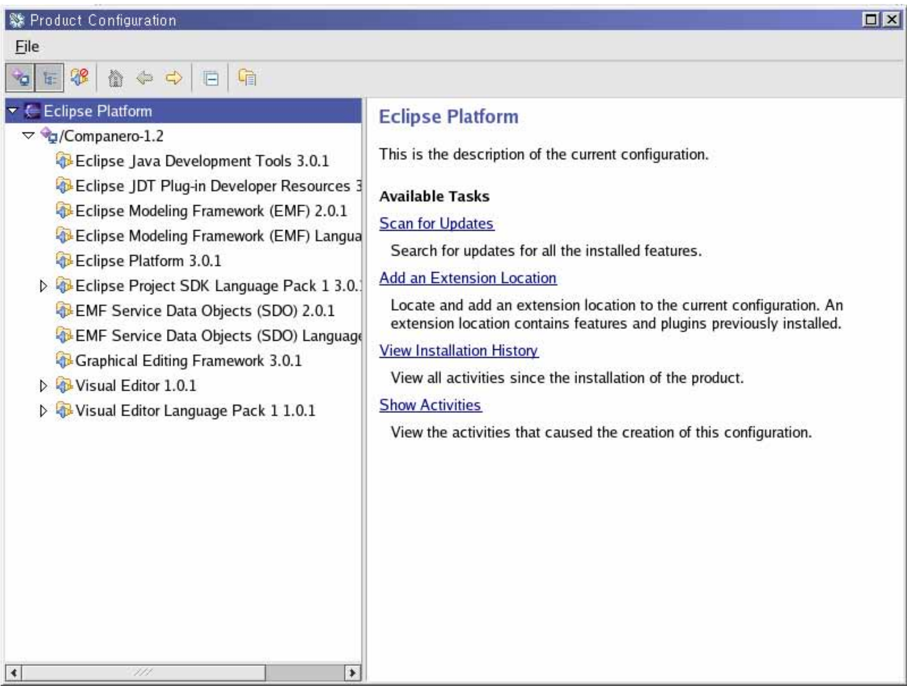


그림 2-23. 끼워넣기구성의 관리

왼쪽에는 Eclipse가 설치되여있는 장소와 끼워넣기의 계층구조가 현시된다. 대상으로 되는 끼워넣기를 선택하면 오른쪽에 판본 등의 자세한 정보가 표시된다. 이 정보안의 [Disable]를 찰칵하면 끼워넣기를 무효로 할수 있다. 사용하지 않는 끼워넣기를 무효로 함으로써 Eclipse의 기동과 동작을 가볍게 할수 있다. 또한 상세정보안의 [Scan for Updates]를 찰칵하면 망경유로 끼워넣기의 갱신거점과 접속하여 새 판본의 확인과 갱신을 할수 있다.

- 설치/갱신조수에 의한 끼워넣기도입

새로 끼워넣기를 설치하는 방법에는 설치/갱신조수를 리용하는 방법이 있다. 이 방법은 끼워넣기의 갱신거점을 지정하기만 하면 망 또는 파일체계경유로 자동적으로 끼워넣기의 판본을 검사하고 끼워넣기를 내리적재하여 설치할수 있다. 끼워넣기의 갱신거점이 있으면 간단히 끼워넣기를 설치할수 있다.

여기서는 Easy Struts의 Easy Explorer끼워넣기의 설치를 실례들어 설명한다.

설치/갱신조수를 기동시키려면 [Help]→[Software Updates]→[Find and Install]를 선택한다. 조수의 첫 화면에서 [Search for features to install]을 선택하고 [Next]단추를 찰칵한다. 그러면 [Install]대화칸이 표시된다. 여기서 끼워넣기의 갱신거점을 추가한다. [New Remote Site]단추를 찰칵하면 명칭과 URL의 입력대화칸이 나오는데 여기서 명칭에는《Easy Struts》, URL에《http://easystruts.sourceforge.net/eclipse/updates/》

을 입력하고 [OK] 단추를 찰칵한다. ▷를 찰칵하여 계층구조를 전개하면 설정이 정확한 조건에서 거점으로부터 끼워넣기의 정보가 얻어지며 그림 2-24와 같이 설치가능한 끼워넣기의 일람이 현시된다. 여기서는 《Easy Explorer plugin》을 선택하고 [Next] 단추를 찰칵한다.

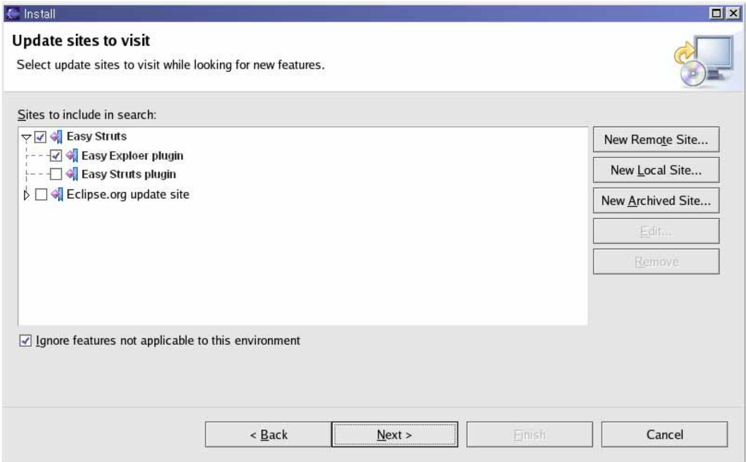


그림 2-24. 끼워넣기의 설치

다음 화면에서는 지정한 끼워넣기별 일람이 표시된다. 이 안에서 설치하려는 끼워넣기에 선택표시를 한다. 여기서는 《Easy Explorer plugin》의 《1.0.1》을 선택하고 [Next]단추를 찰칵한다. 다음 허가사항이 표시되는데 [I accept the agreement]를 선택하고 [Next]단추를 찰칵한다.

다음 화면에서 지정한 끼워넣기의 정보와 끼워넣기의 설치위치를 확인하고 [Finish]단추를 찰칵하면 끼워넣기의 설치가 시작된다. 마지막으로 Eclipse의 재기동을 독촉하는 대화칸이 나오는데 [Yes]단추를 찰칵하면 Eclipse가 재기동된다(그림 2-25).

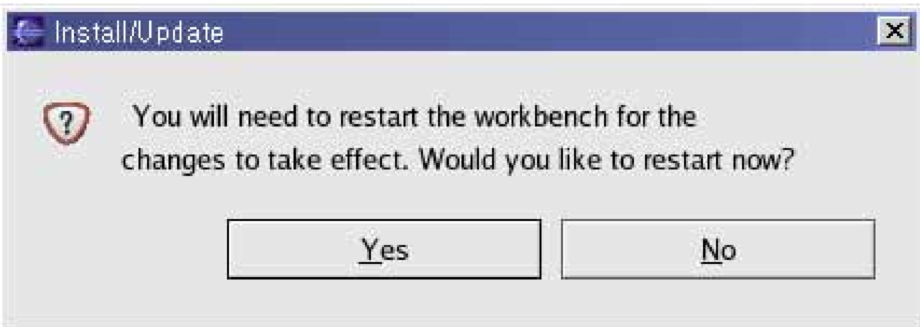


그림 2-25. 설치후의 재기동대화칸

실지로 설치한 끼워넣기를 리용해보기로 한다. [Navigator]보기에서 파일 또는 등록부를 선택한 다음 지름차림표를 열면 [Easy Explorer]항목이 있는것을 알수 있다. 이것을 선택하면 선택되여있는 등록부가 열람기에 열린다. 이처럼 EasyExplorer가 설치되여 기능이 추가된것을 확인할수 있다.

이와 같이 갱신조수에 의한 설치에서는 URL를 지정하고 화면에 따라 조작하기만하면 간단히 끼워넣기의 설치를 할수 있다. 또한 내리적재한 끼워넣기파일이 site.xml를 포함하고있을 때에는 설치대화칸의 [New Archive Site] 또는 [New Local Site]에서 자기 콤퓨터의 파일을 지정하여 설치할수도 있다.

대리봉사(proxy)를 경유하는 끼워넣기의 갱신

기업내국부망에 접속되여있고 대리봉사로 인터네트에 접근하는 환경에서는 끼워넣기갱신의 대리설정이 필요하다. 이러한 경우에는 차림표띠에서 [Window]→[Preferences]→[Install/Update)]을 선택하고 오른쪽에 현시되는 [Proxy settings]에서 [Enable HTTP proxy connection]검사칸을 선택한다. 그리고 [HTTP proxy host address]에는 대리봉사기의 IP주소를 주고 [HTTP proxy host port]에는 대리봉사기의 포구번호를 설정한다(그림 2-26).

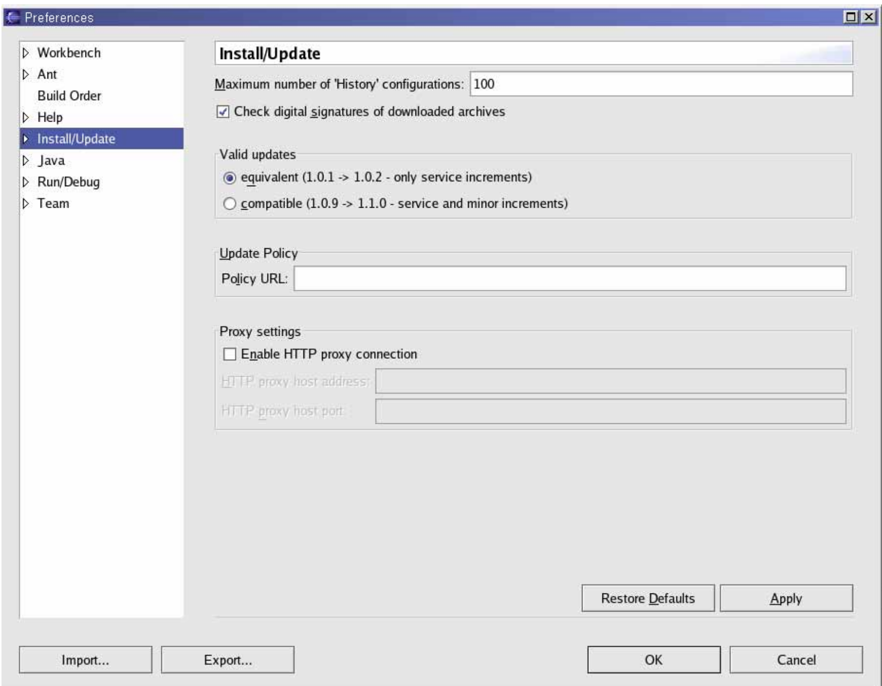


그림 2-26. 대리봉사기의 설정

2.4.2. 수동설치방법

갱신조수에 의한 설치외에 끼워넣기의 파일을 수동으로 복사하여 설치하는 방법이 있다. 조수설치방법을 지원하지 않는 끼워넣기는 수동으로 설치한다. 여기서는 eclipse.org안의 Eclipse Tools Project에서 개발되고있는 Visual Editor(VE)끼워넣기에 대하여 설명한다.

수동설치방법은 다음과 같은 순서로 진행한다.

1. Eclipse가 기동되여있는 경우 일단 완료한다.
2. eclipse.org의 EMF프로젝트(http://www.eclipse.org/emf)로부터 EMF, SDO의 실행파일(emf-sdo-runtime-2.0.1.zip)을 내리적재한다
3. 2에서 얻은 파일을 풀면 eclipse라는 등록부아래에 feature와plugins라는 등록부가 생긴다. feature와 plugins등록부를 《/eclipse》안으로 이동한다(보존파일을 직접 《/》아래에 덧쓰기방식으로 풀어도 된다).
4. eclipse.org의 GEF프로젝트(http://www.eclipse.org/emf)로부터 GEF의 실행파일(GEF-runtime-3.0.1.zip)를 입수하고 3.에서처럼 풀어서 생긴 feature와 plugins등록부의 내용을 《/eclipse》에 덧쓰기이동한다.
5. eclipse.org의 Visual Editor프로젝트(http://www.eclipse.org/vep/)로부터 Virtual Editor의 실행파일 (VE-runtime-1.0.zip)를 입수하고 3.에서처럼 풀어서 생긴 feature와 plugins등록부의 내용을《/eclipse》에 덧쓰기이동한다.
6. EMF, SDO, GEF, VE의 언어패키지(Nlpack-emf-sdo-runtime-2.0.0.zip, Nlpack-GEF-runtime-3.0.1.zip, Nlpack-VE-runtime-1.0.x.zip)파일을 입수하고 3.에서처럼 풀어놓은 feature와plugins등록부의 내용을 《/eclipse》에 덧쓰기이동한다.
7. Eclipse를 재기동한다.

등록부의 이름을 보면 알수 있는바와 같이 끼워넣기파일은 《[eclipse가 있는 등록부]/plugins/[끼워넣기패키지이름]_<판본번호>》라는 등록부에 놓인다. 마찬가지로 특성정보(끼워넣기를 모아놓은것)에 관한 파일들은 《 [eclipse가 있는 등록부]/features/[끼워넣기패키지이름]_<판본번호>》라는 등록부에 놓인다. 끼워넣기와 특성정보의 설치는 해당등록부에 파일을 놓기만하면 끝난다. Eclipse는 기동할 때에 새로운 끼워넣기등록부가 있는가를 검사하고 파일이 있으면 자동적으로 끼워넣기 및 특성정보를 읽어들인다.

VE끼워넣기의 설치가 성공하면 편집기로 Visual Editor가 추가된다. Swing 및 SWT의 도형사용자대면부요소의 파일을 열면 도형적으로 표시 및 편집할수 있다(그림 2-27).

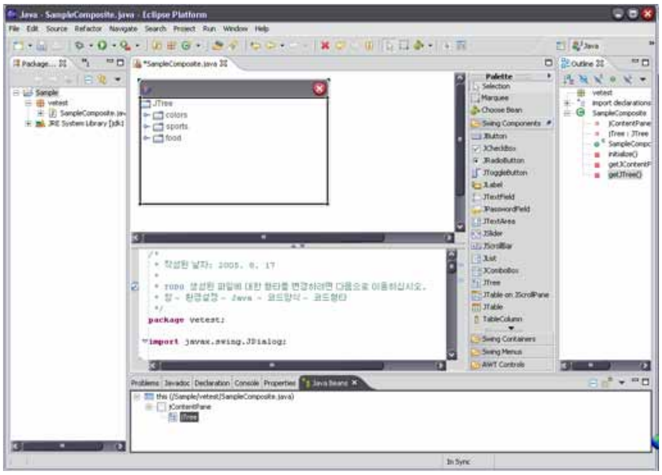


그림 2-27. Visual Editor에 의한 GUI요소의 편집

이전 끼워넣기와의 호환성

Eclipse 3.x와 Eclipse 2.x사이에서의 끼워넣기호환성은 완전히 보장되여있지 않다. 그러나 기본적인 부분은 같기때문에 약간만 수정해도 되는 끼워넣기도 많다. 특히 간단한 기능의 끼워넣기는 정식으로 Eclipse 3.0에 대응되여있지 않아도 움직이는 경우도 있다(《Eclipse Explorer plugin》이 그러한 실례로 된다).

Eclipse 2.x에서는 JDK1.3에 기초하고 XML구문분석기(parser)가 JDK에 포함되여있지 않으므로 SDK의 내부에 org.apache.xerces라는 끼워넣기가 있었다. 그러나 eclipse3.0부터는 JDK1.4기반이므로 JDK에 XML구문분석기가 포함되여있어 xerces끼워넣기는 생략되였다. 그 결과 xerces끼워넣기를 리용하고있는것은 그대로는 Eclipse 3.0에서 쓸수 없다. 그러나 아래와 같이 하면 Eclipse 2.x끼워넣기가 높은 판본에서 동작하는 경우가 있다.

1. Eclipse 3.0의 plugins등록부에 Eclipse 2.x부터 org.apache.xerces_x.x.x의 등록부를 복사한다.
2. 해당 끼워넣기의 plugin.xml에 《"<import plugin="org. apache.xerces" />"》가 존재하는 경우에는 삭제한다.

이 방법으로 x-men(ver.0.4.2.3) 등이 동작할수 있게 된다.

2.4.4. 끼워넣기의 판본갱신

끼워넣기의 판본을 최신으로 갱신하자면 [Help]→[Software Updates]→[Find and Install]를 선택하면 현시되는 대화칸을 리용한다. 조수의 첫 화면에서 [Search for updates of the currently installed features]을 선택한다. 현재 설치되여있는 끼워넣기의 갱신정보를 검색하고 설치할수 있다. 끼워넣기의 갱신은 [Window]→[Preferences]→[Install/Update]→[Automatic Updates]을 통하여서도 할수 있다. 이와 같은 설정은 기동할 때에 혹은 일정한 간격으로 끼워넣기를 갱신할수 있게 하여준다.

제3장. Eclipse에 의한 Java프로그람작성

제1절. 새로운 프로젝트와 클라스의 작성

3.1.1. 견본프로그람의 개요

3장에서는 간단한 견본프로젝트를 작성하면서 Eclipse상에서의 기본적인 개발절차에 대하여 구체적으로 설명한다. 종업원번호, 이름, 전화번호, 책임자에 관한 정보를 표시하는 Java응용프로그람을 실례들어 설명한다. 작성하려는 견본프로그람의 클라스도를 그림 3-1에 주었다.

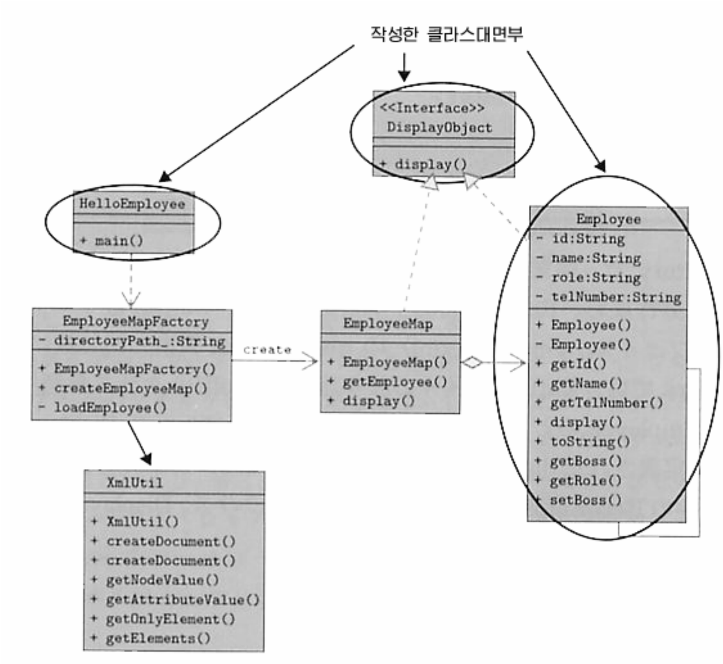


그림 3-1. 견본프로그람의 클라스도

HelloEmployee클라스에서 XML파일을 지정하고 EmployeeMapFactory클라스의 구체례(instance)를 생성한다. 목록 3-1에 읽어들일 XML파일 employee.xml에 대하여 서술하였다. 여기서 매 종업원들은 ID값(id속성)을 가지며 이름(name요소), 직위(role요소), 전화번호(telNumber요소), 책임자의 식별부호(bossId요소)를 가진다.

목록 3-1. employee.xml

```
<?xml version= '1.0' encoding= "Shift_JIS" ?>
<EmployeeList>
    <Employee id= "01" >
        <name>kim kim</name>
        <role>Manager</role>
        <telNumber>3581111</telNumber>
        <bossId/>
    </Employee>
    <Employee id= "02" >
        <name>Li kim</name>
        <role>Designer</role>
        <telNumber>3582222</telNumber>
        <bossId>01<bossId/>
    </Employee>
    <Employee id= "03" >
        <name>Pak kim</name>
        <role>Engineer</role>
        <telNumber>3583333</telNumber>
        <bossId>01<bossId/>
    </Employee>
</EmployeeList>
```

EmployeeMapFactory에서 종업원속성을 표현하는 JavaBean인 Employee클라스의 구체례를 종업원수만큼 작성하여 EmployeeMap에 넣는다. Employee클라스의 실제개수는 읽어들이는 XML파일(employee.xml)의 <employee>요소개수와 같다. 또한 EmployeeMapFactory에서는 XML파일을 읽어들이기 위해 XmlUtil클라스를 리용한다.

EmployeeMap와 Employee클라스들은 모두 대면 DisplayObject를 실현하고 대면에서 정의된 display메쏘드를 리용하여 표준입력에 의한 종업원속성을 출력한다.

3장에서는 HelloEmployee클라스, Employee클라스, DisplayObject대면과 employee.xml파일을 작성한다.

3.1.2. 프로젝트란

Eclipse로 Java프로그람작성을 시작하기 전에 우선 프로젝트를 작성하여야 한다. **프로젝트란** 원천코드나 생성되는 class파일, 클라스경로의 설정 등을 묶어서 관리하기 위한 단위를 말한다. Eclipse에서는 2개이상의 프로젝트를 동시에 작업공간에 열어놓을 수 있다.

3.1.3. 프로젝트의 작성

Eclipse를 기동시켜 새로 프로젝트를 작성하는 절차는 다음과 같다.

차림표띠에서 [File]→[New]→[Project]를 선택한다. 그러면 새 프로젝트를 작성하기 위한 대화칸이 현시된다(그림 3-2).

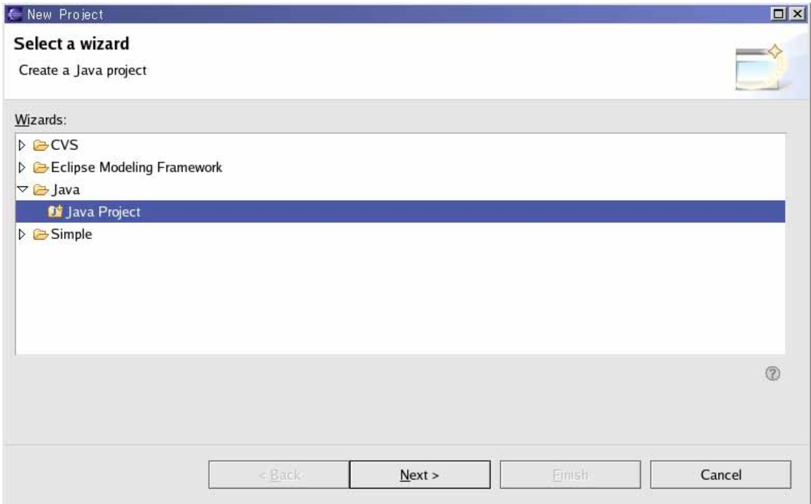


그림 3-2. 새로운 프로젝트작성대화칸

[Java Project]를 선택하고 [Next]단추를 찰칵한다.

이때 현시되는 대화칸에서 프로젝트이름과 보존등록부, 프로젝트안의 원천코드와 class파일의 등록부구성을 지정한다. [Project name]에 《SampleChap3》라고 입력한다. 프로젝트를 보존하는 [Location]에서 기정의 [Create project in workspace]을 선택함으로써 2장에서 지정한 작업공간인 《/kut/workspace》를 사용한다. [Project layout]는 [Create separate source and output folders]을 선택한다. 이렇게하면 Java원천코드는 프로젝트등록부아래의 src등록부에 들어가며 class파일은 bin등록부아래에 생성된다(그림 3-3).

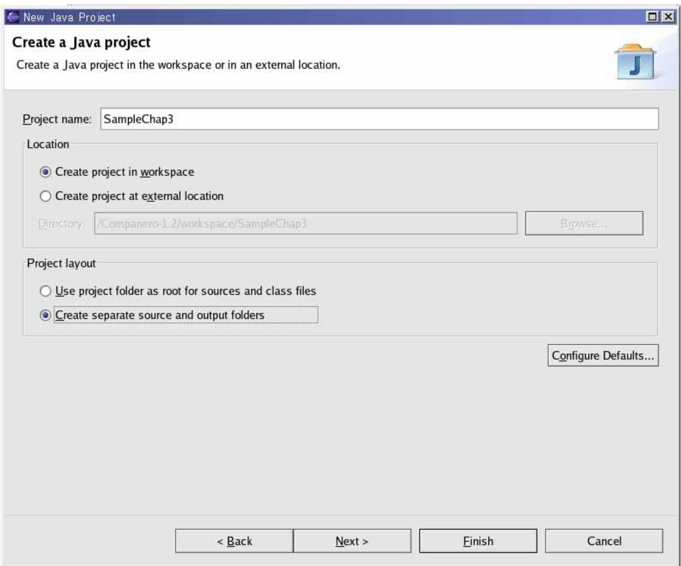


그림 3-3. 프로젝트이름의 입력

여기서 [Configure Defaults]단추를 찰칵하고 [Source and output folder]에서 [Folders]를 선택한다(그림 3-4).

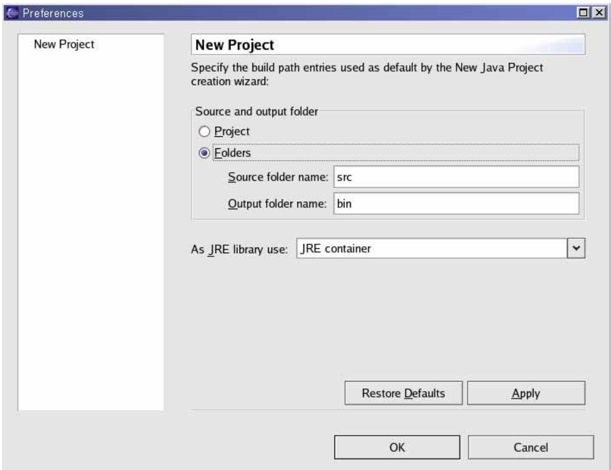


그림 3-4. 새 프로젝트에서 사용할 경로설정

새 프로젝트작성대화칸의 [Finish]단추를 찰칵하면 Java투시가 자동적으로 추가되고 그림 3-5와 같은 화면이 나타난다. 이상으로 Java프로그람을 작성하기 위한 준비가 완성된다.

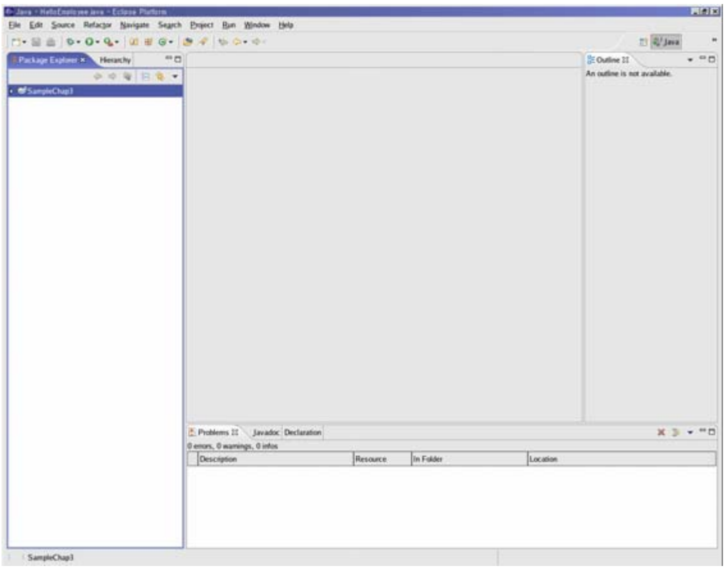


그림 3-5. 새로 생성된 프로젝트

행번호의 표시

기정설정으로는 Java편집기의 행번호가 현시되지 않는다. 행번호를 참조할 때가 많기때문에 기정설정을 변경하여 현시되도록 하는것이 좋다. 차림표띠에서 [Window]→[Preferences]을 선택하여 나타나는 대화칸의 왼쪽에서 [Java]→[Editor]를 선택한다. [Appearance]태브를 눌러 현시되는 대화칸에서 [Show line numbers]를 선택한다.

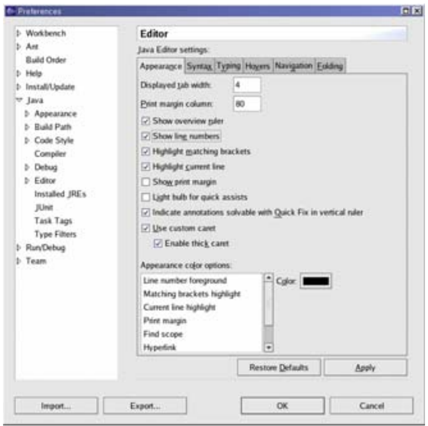


그림 3-6. 행번호의 표시

3.1.4. 프로젝트속성(project property)

여기서는 4장에서 보게 되는 Junit를 리용하여 클라스경로의 설정을 진행한다. 앞으로의 판본갱신에 대한 확장성을 보장하기 위하여 일단 클라스경로를 junit.jar로 설정하고 그 클라스경로를 프로젝트의 건립경로로 설정한다. 구체적인 절차는 다음과 같다.

클라스경로변수의 설정이나 편집은 차림표띠의 [Window]→[Preferences]에서 진행한다. 여기서 이미 정의되여있는 클라스경로변수는 [JUNIT_HOME]이라는것을 알수

있다. [JUNIT_HOME]을 선택한 상태에서 [Edit]를 찰칵하여 현시되는 대화칸에서 보면 [Name]에는 《JUNIT_HOME》, [Path]에는 《/kut/plugins/org.junit_3.8.1/》이 입력되여있는것을 확인할수 있다(그림 3-7).

여기서 이 클라스경로변수 [JUNIT_HOME]을 그대로 리용하기로 하고 확인이 끝나면 [Cancel]단추를 찰칵한다.

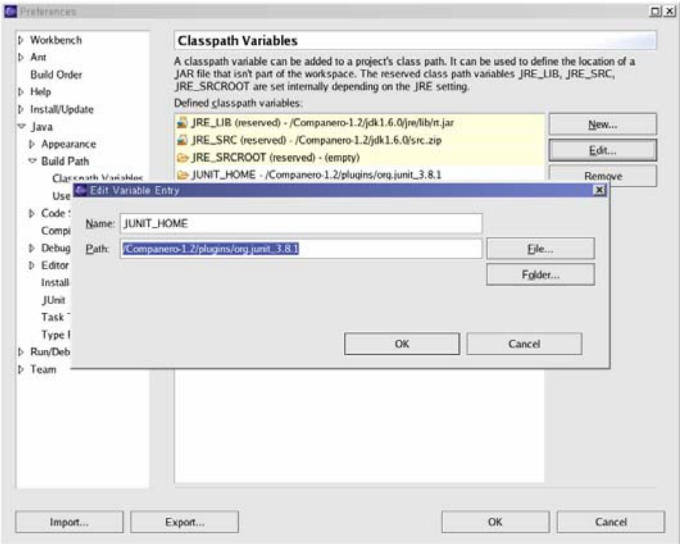


그림 3-7. 클라스경로변수의 확인

이 조작에 의하여 현재 프로젝트에서는 junit.jar에 대하여 클라스경로가 지정된다.

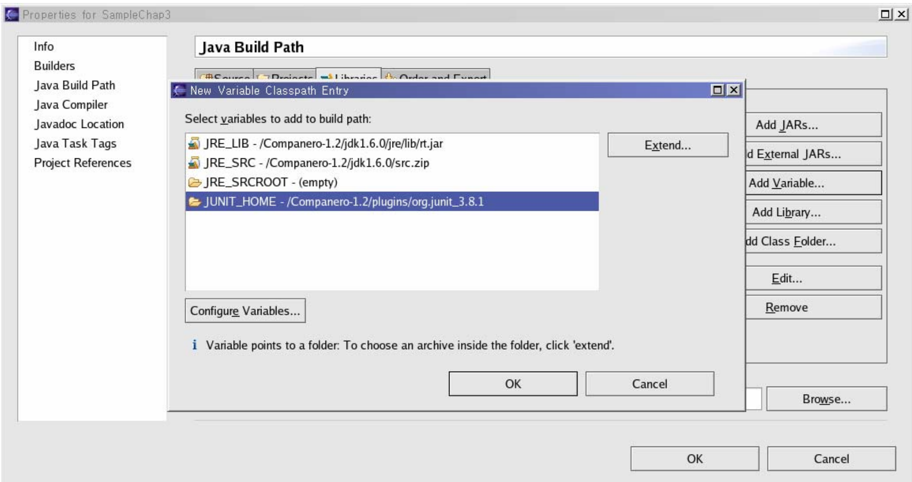


그림 3-8. Java건립경로의 추가

추가한 서고에 원천코드와 Javadoc를 부가

미리 설정되여있는 클라스경로변수 JUNIT_HOME이 참조하고있는것처럼 junit.jar는 Eclipse의 끼워넣기등록부(/kut/eclipse/ plugins/orgjunit_3.8.1)에 포함되여있는데 junit.jar에 원천과 Javadoc를 부가하기 위해서는 Junit의 원천과 Javadoc가 필요하게 된다.

그래서 Junit의 압축파일 junit3.8.1을 복사하고 풀면 junit.jar와 함께 src.jar와 Javadoc등록부도 작성된다. 이 파일들을 《/kut/junit/》아래에 보존한다. 여기서 JUNIT_HOME에 Junit의 원천인 src.jar를 관련시킴으로써 Eclipse가 Junit의 원천코드를 참조할수 있게 된다. 례하면 ctrl건을 누르면서 junit.jar에 포함되여있는 클라스나 메쏘드우에 마우스유표를 맞추어 클릭하면 해당한 원천코드에로 뛰여넘기할수 있다.

또한 마우스유표를 메쏘드나 클라스에 맞추면 원천코드안에 서술되여있는 Javadoc해설문이 튀여나오기도움말(pop-up)로 표시된다. 또한 내용물방조에도 원천코드안의 Javadoc해설문이 표시된다(［3.2.12 해설문, 해설문해제와 풍선도움말］참고).

src.jar의 경로를 설정하는것도 junit.jar의 경우처럼 먼저 차림표띠로부터 [Window]→[Preferences]을 선택하여 표시되는 대화칸의 [Java], [Build Path]를 전개하고 [Classpath Variable]를 선택한다. [New]를 찰칵하면 클라스경로변수 [JUNIT_SRC]를 작성한다. 경로는 압축을 푼 후 src2.jar의 장소인 c/ASCii/junit/src.jar를 지정한다(그림 3-9).

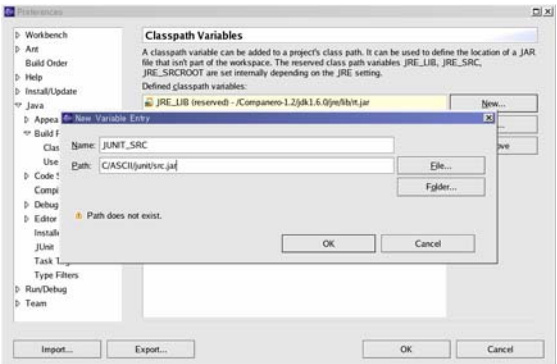


그림 3-9. 클라스경로변수 《JUNIT_SRC》의 작성

[Package Exproler]보기에서 프로젝트 SampleChap3을 선택한 상태에서 차림표띠의 [File]→[Properties]로부터 [Properties for SampleChap3]대화칸을 연다. 왼쪽화면에서 [Java Build Path]를 선택하고 [Libraries]태브를 누른다. 앞에

서 추가한 [JUNIT_HOME]를 전개하면 [Source attachment]가 표시되는데 이것을 선택하고 [Edit]를 찰칵한다. 이때 현시되는 [Source Attachment Configuration]에서 [Variable]를 찰칵하면 [Variable Selection]대화칸이 현시된다. 여기서 클라스경로변수 JUNIT_SRC를 선택하고 [OK]를 찰칵한다(그림 3-10). 이것으로 설정은 끝난다.

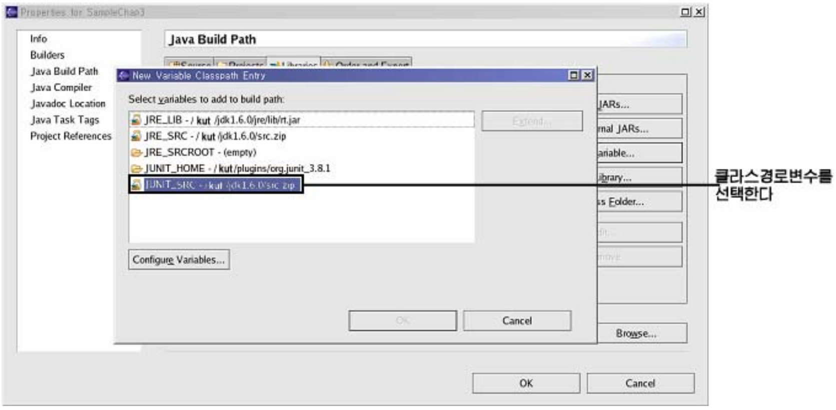


그림 3-10. 클라스경로변수 《JUNIT_SRC》의 선택

마찬가지로 [JUNIT_HOME]를 전개하면 표시되는 [Javadoc location]를 편집하고 Junit를 압축을 풀 때 생성되는 Javadoc의 등록부를 지정한다. Javadoc의 위치는 클라스변수경로가 아니라 실지 등록부인 《/kut/juint/javadoc》를 지정한다. 이 조작에 의하여 Junit의 Javadoc내용이 Eclipse의 도움말로부터 참조할수 있게 된다.

3.1.5. 클라스의 작성

계속하여 작성한 SampleChap3프로젝트에 클라스파일을 작성한다.

먼저 HelloEmployee.java를 작성한다. [Package Explorer]보기에서 [src]등록부를 선택한 다음 지름차림표를 열고 [New]→[Class]를 선택한다.

차림표띠의 [File]→[New]→[Class]로서도 같은 조작을 할수 있다. 현시되는 대화칸에서 새로 작성하는 Java클라스의 설정을 진행한다. [Package]에 《sample.hello》, [Name]에는 《HelloEmployee》를, [Modifiers]에는 [public]를 지정한다. 여기서는 [public static void main(String[] args)]검사칸을 선택하여 정적클라스를 작성하도록 하고있다(그림 3-11).

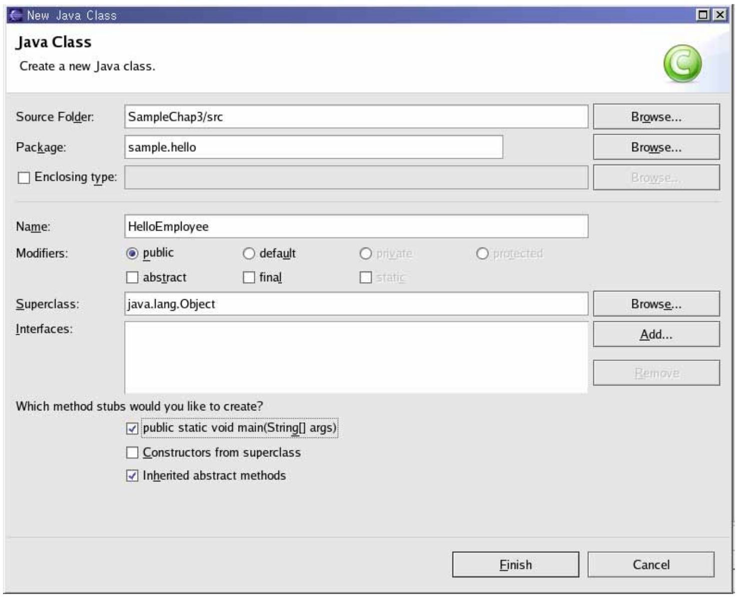


그림 3-11. 새 클라스의 작성

[Finish]단추를 찰칵하면 패키지와 클라스이름, main메쏘드가 이미 정의된 HelloEmployee.java가 Java편집기에 현시된다(그림 3-12).

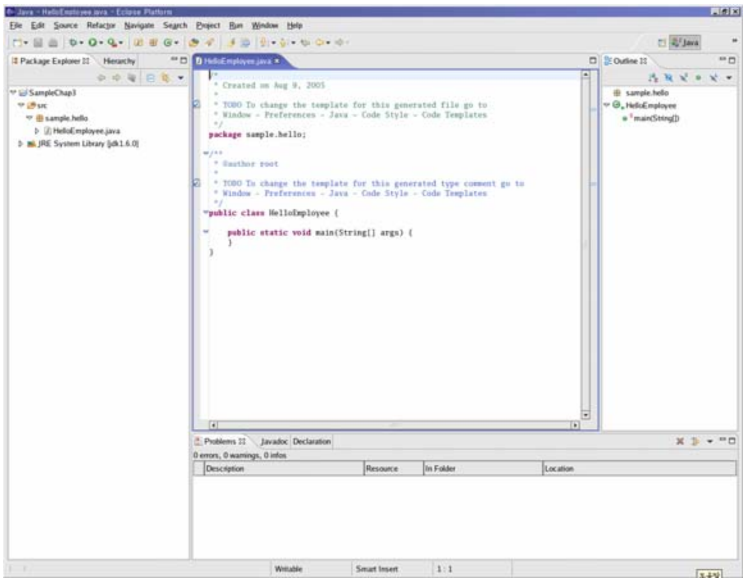


그림 3-12. 생성된 클라스파일

제2절. Java편집기의 사용법

여기서는 견본코드를 작성해보면서 Eclipse에서 Java편집기의 사용방법에 대하여 설명하기로 한다. 본문편집기 등을 사용하여 코드를 작성할 때 보다 훨씬 개발효률이 높아진다는것을 알수 있다.

3.2.1. 메쏘드와 괄호의 자동설정

먼저 간단한 System.out.prinln를 리용하여 《Hello World》를 출력하는 실례에 대하여 보기로 한다.

앞서 작성한 HelloEmployee.java의 main메쏘드안에 《System.》라고 입력한 다음 잠간 기다리면 그림 3-13와 같이 System클라스에 포함되는 err, in, out, ...라는 마당(field)이나 메쏘드의 후보가 표시된다. 이것을 내용물방조라고 부른다. 유표를 아래우로 이동시켜 후보중에서 하나를 선택하고 Enter건을 누르면 선택한 마당이 자동입력된다. 또한 입력하려는 마당이름인 out의 첫글자《o》를 건입력하는 방법으로 후보범위를 줄일수도 있다.

Println메쏘드의 입력도 마찬가지이다. 《.》을 입력하고 잠간 기다리면 마당 System.out의 PrintStream클라스에 포함되는 메쏘드들이 입력후보로 표시된다. 많은 후보가 표시되기때문에 《println》을 건입력하여 차례로 후보를 줄여간다. 지름건 Ctrl + Space를 누르면 내용물방조를 곧 표시할수도 있다. println메쏘드의 인수에 《Hello World》를 입력한다. 첫 인용부호를 입력하면 이것을 닫기 위한 인용부호가 자동적으로 입력되는것을 알수 있다. 이것도 Eclipe에 의한 방조기능중의 하나로서 괄호의 자동설정이라고 한다. 인용괄호외에도 메쏘드의 중괄호나 인수의 소괄호 등도 자동입력되기때문에 괄호닫기를 잊어버리는 오유를 방지할수 있다.

```
public static void main(String[] args) {
System.out.println( “Hello World” );
 }
```

입력이 끝나면 차림표띠의 [File]→[Save]에서 또는 지름건 Ctrl + S를 리용하여 원천코드를 보관한다. 기정설정에서는 이 시점에서 자동적으로 콤파일이 진행되고 bin 등록부아래에 class파일이 생성된다.

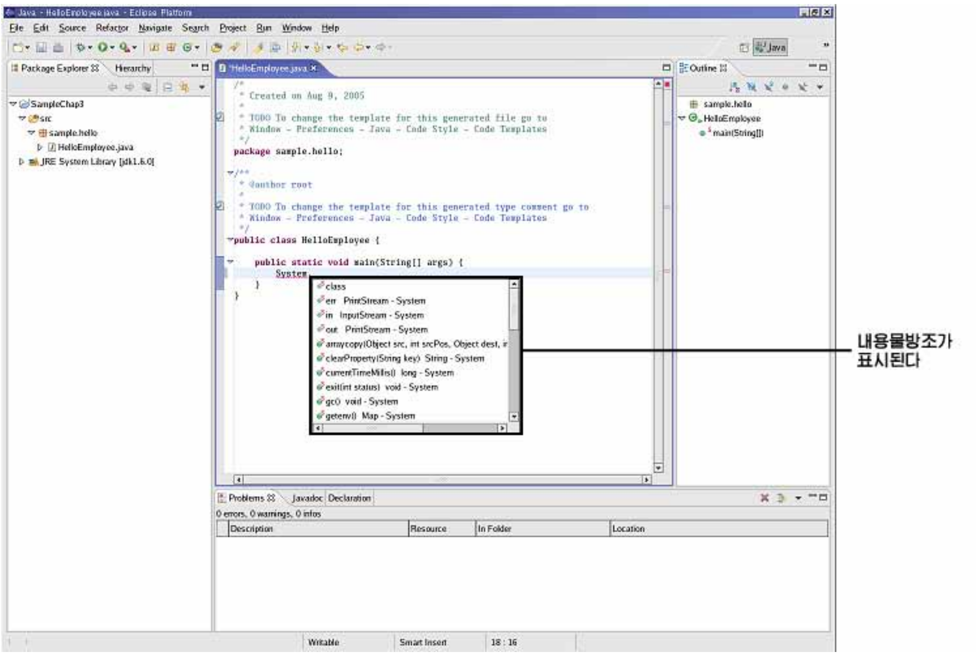


그림 3-13. 메쏘드나 마당의 빠른 선택

건 립

Eclipse의 기정설정에서는 보통 원천코드를 보존하기만하면 건립된다. 그것은 차림표띠의 [Window]→[Preferences]를 선택하여 현시되는 대화칸에서 [Workbench]를 선택했을 때 [Build automatically]검사칸이 선택상태로 되여있기때문이다. 처리부하가 크다는 리유로 이 선택표시를 해제시킨 경우에는 차림표띠에서 [Project]→[Build Project] 또는 [Build All]을 선택하여 건립한다. 지름건 Ctrl + B를 리용해도 된다.

3.2.2. 프로그람의 실행

《 Hello World 》를 실행하기로 한다. [Package Explorer]보기에서 HelloEmployee.java가 선택되여있는가를 확인한 다음 차림표띠에서 [Run]→[Run AS]→[Java Application]을 찰칵한다. 그러면 선택한 클라스가 실행된다. 표준설정에서는 Java편집기의 아래에 있는 [Console]보기에 표준출력으로 《Hello World》가 표시된다(그림 3-14).

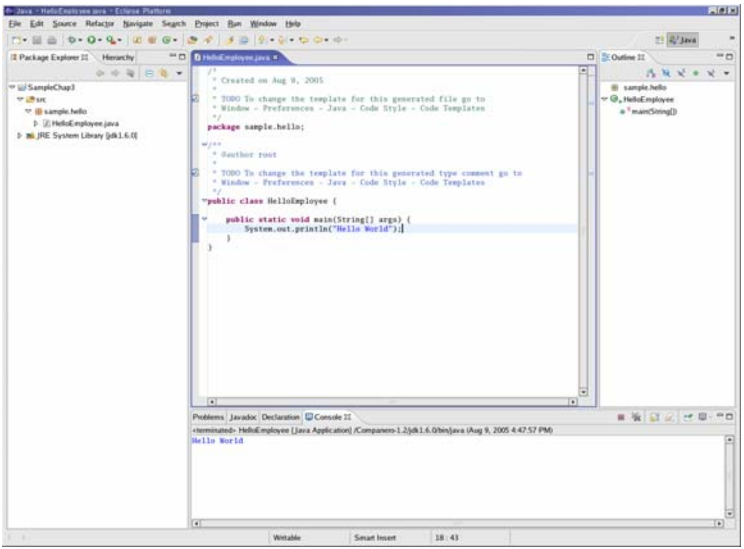


그림 3-14. Hello World의 실행

3.2.3. 보기 쉬운 원천코드의 작성

Eclipse에는 원천코드를 보기 쉽게 하기 위한 코드서식기능, Javadoc해설문의 추가기능 등이 있다.

Java편집기상에서 지름차림표의 [Source]→[Format]를 선택함으로써 원천코드의 들여쓰기, 괄호의 위치맞추기 등을 할수 있다. 범위를 선택한 상태에서 서식화를 실행하면 선택된 부분에 대하여만 서식화가 적용된다.

서식화의 정의는 차림표띠에서 [Window]→[Preferences]을 선택하면 현시되는 대화칸의 [Java]→[Code Style]→[Code Formatter]에서 진행한다(그림 3-15). 여기서 [New]를 찰칵하고 기존 외형(profile)에 기초하여 서식을 정의한다. 행바꾸기, 한 행당 문자수, 들여쓰기태브의 개수, 공백을 넣을자리 등을 필요에 따라 또는 해당 프로젝트의 개발방향에 따라 설정한다.

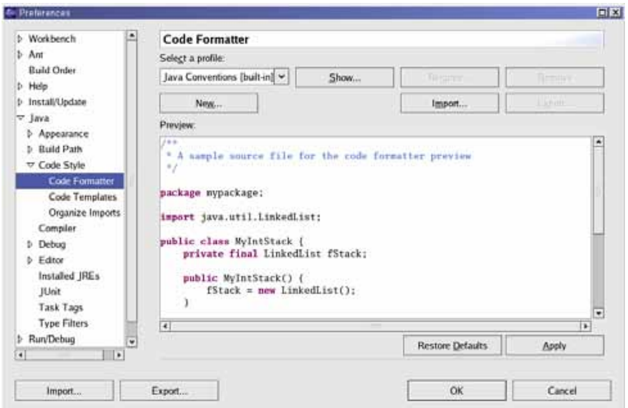


그림 3-15. 코드서식기

사용자가 정의한 외형은 xml형식으로 보존하거나 읽어들일수 있으므로 개발조안에서 설정을 공유할 때 편리하다. 보다 구체적인것은 [6.7.1. 코드서식기]에서 보기로 한다.

Javadoc용을 비롯한 해설문을 추가할 때도 그 일부를 자동적으로 삽입할수 있다. Java클라스나 대면을 새로 작성하면 앞머리에 파일해설문이 삽입된다. 기정설정에서는 코드를 생성한 날자가 삽입되게 되여있는데 설정을 변경하여 기업소이름, 저작권(copy right), CVS용태그 등을 삽입하도록 정의할수 있다.

차림표띠의 [Window]→[Preferences]을 찰칵하면 현시되는 [Preferences]대화칸에서 [Java]→[Code Style]→[Code Templates]를 선택한다. 화면오른쪽의 [Configure generated code and comments]에서 [Code]를 전개한다. [New Java files]을 선택한 상태에서 [Edit]단추를 찰칵하면 원형의 편집이 가능하게 된다(그림 3-16).

메쏘드에 대한 Javadoc해설문은 메쏘드안에서 지름차림표를 열고[Source]→[Add Javadoc Comment]를 선택하면 삽입된다. 메쏘드의 밖에서 동일한 조작을 진행하면 클라스에 대한 Javadoc해설문이 추가된다. 변경하려고할 때에는 파일앞머리에 삽입되는 파일해설문과 마찬가지로 차림표띠의 [Window]→[Preferences]를 선택하면 현시되는 대화칸에서 [Java]→[Code Style]→[Code Templates]를 선택한다.

메쏘드에 대한 Javadoc해설문은 [Comments]→[Method]에서, 그리고 클라스에 대한 Javadoc는 [Comments]→[Types]에서 편집한다. 이 대화칸의 [Automatically add comments for new methods and types]검사칸의 선택표시를 해제함으로써 해설문을 자동적으로 삽입하지 않도록 할수도 있다.

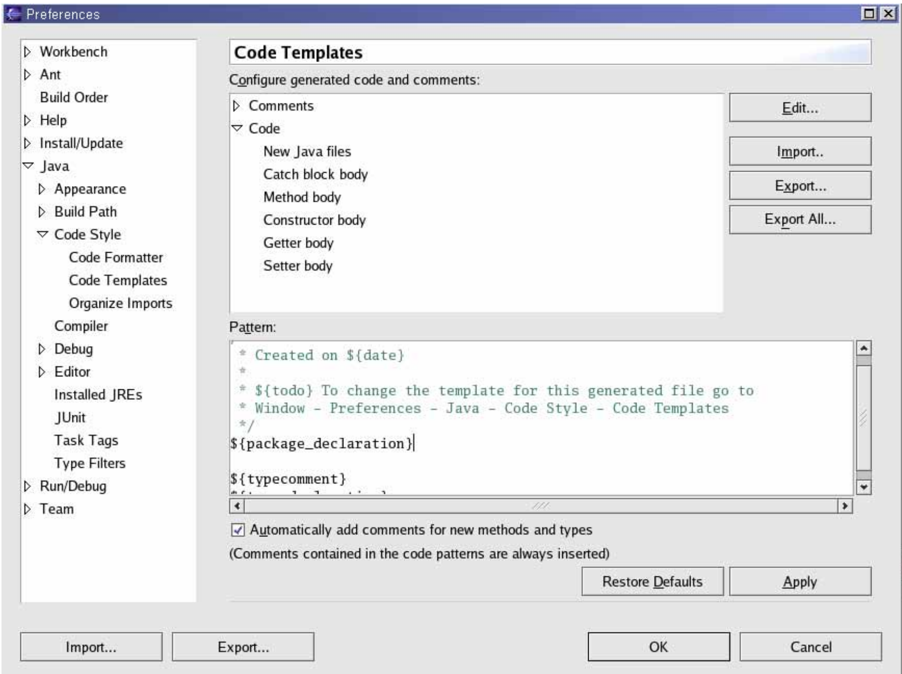


그림 3-16. 코드의 형판

코드와 해설문의 편집은 편집대상을 선택한 다음 [Edit]단추를 찰칵하여 진행한다. [Insert Variable]을 리용하면 사용자명과 현재날자, CVS태그에 리용할수 있는 $기호 등 미리 준비되여있는 변수를 유표의 위치에 넣을수 있다(그림 3-17).

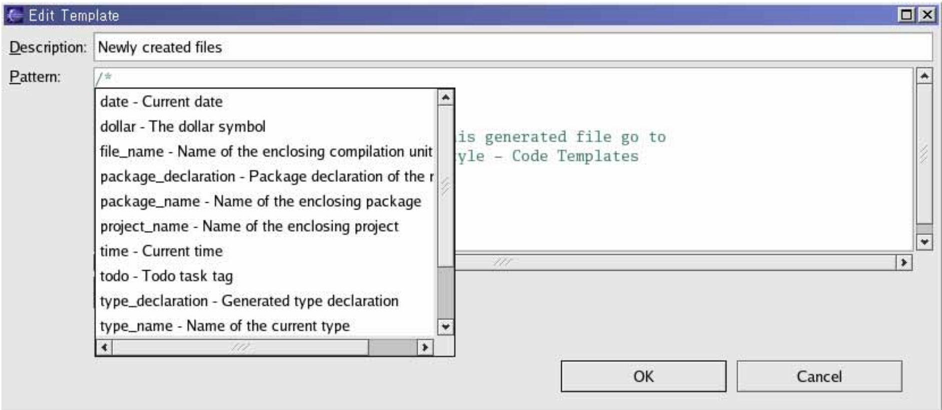


그림 3-17. 변수의 삽입

3.2.4. [Tasks]보기와 [Problems]보기

새 파일과 클라스가 작성될 때 TODO과제가 추가된다. 보기 쉬운 코드가 되도록 하기 위하여 TODO부분을 될수록 해설문으로 바꿔놓도록 한다.

해설문에 대하여서만이 아니라 Eclipse가 자동적으로 추가한 추상메쏘드의 대용체(stub), 원천의 자동입력기능으로 추가된 try/catch의 catch문 등도 TODO과제의 대상으로 되여있다. 이것은 Eclipse가 자동적으로 생성한 원천코드에 대하여 후에 사용자가 내용을 넣어 줄 필요가 있기때문에 잊지 않도록 TODO일람에 추가하여 관리하는것이다.

TODO과제일람은 [Tasks]보기에서 참조할수 있다.

[Tasks]보기는 차림표띠의 [Window]→[Show View]→[Other]를 선택하면 현시되는 대화칸에서 [Basic]등록부를 전개하고 [Tasks]를 선택하면 추가된다. 과제는 사용자가 임의의 행에 대하여 추가할수도 있다. 추가하려는 행의 왼쪽 재색부분에서 지름차림표의 [Add Task]를 선택한다. 또는 추가하려는 행에 유표가 놓인 상태에서 차림표띠의 [Edit]→[Add Task]를 선택해도 된다. 이것은 현시점에서는 명세서가 결정되여있지 않지만 명세서가 결정된 후에 추가하려고 생각하고있는 부분 등에 TODO과제를 설정해둠으로써 코드넣기작업을 잊어버리는 오유를 방지할수 있다. 사용자가 추가한 과제에 대해서는 완료표시를 하거나 삭제하며 [Tasks]보기안의 완료표시가 있는 과제만을 골라 삭제하는것과 같은 작업을 진행할수 있다.

Eclipse가 알아낸 콤파일오유와 경고 등은 [Problems]보기에 자동적으로 추가되고 표시된다. 매개 보기의 항목을 두번찰칵하면 다른 클라스가 열려져있어도 작업대상으로 되고있는 클라스의 해당위치(행)에 뛰여넘기할수 있다.

3.2.5. 대면의 작성

견본프로그람에서 리용하는 현시용객체의 대면 DisplsyObject.Java를 작성한다. 패키지 sample.hello에 작성하기때문에 [Package Explorer]보기에서 패키지를 선택한 다음 지름차림표를 열거나 또는 차림표띠의 [File]에서 [New]→[Interface]를 찰칵한다. 추가하려는 패키지를 선택하고 새로운 대면작성대화칸을 연다. 이 대화칸의 패키지명에 선택한 패키지가 기정으로 표시되게 된다(그림 3-18).

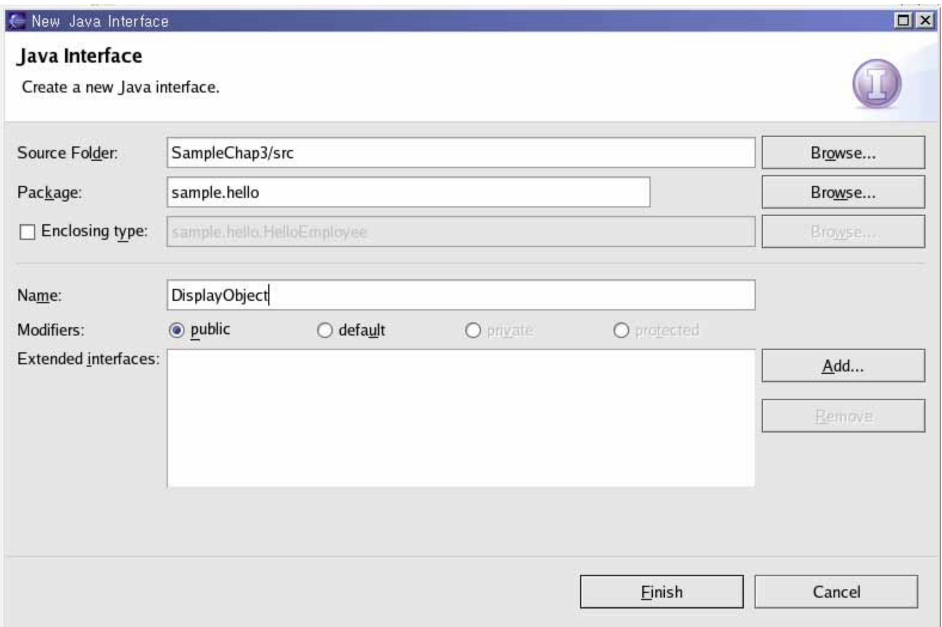


그림 3-18. 새로운 대면의 작성

이름을 DisplyObject라고 하고 대화칸을 닫으면 대면클라스가 생성된다. 여기서 DisplayObject대면은 display메쏘드를 가지고있는것으로 본다. Void display()를 아래와 같이 정의한다. Javadoc주석도 써놓는다.

```
package sample.hello;
public interface DisplayObject {
 /**
* 얻은 결과를 출력한다.
*/
void display();
  }
```

그리고 DisplayObject.java를 보존한다.

3.2.6. 대면의 실현

우에서 작성한 대면 DisplayObject를 추가한 Employee.java를 작성한다. 이 클라스는 XML파일로부터 읽어들인 종업원에 관한 속성을 보유하는 JavaBean이다.

대면을 새로 작성할 때와 마찬가지로 sample.hello패키지를 선택한 다음에 클라스의 새로 작성을 진행하면 패키지명이 미리 입력된 대화칸이 현시된다. [Name]에 Java 클라스이름으로 《Employee》를 입력한다. 계속하여 [Interface]의 [Add]를 찰칵하면 [Implemented Interfaces Selection]대화칸이 현시된다. 추가하려는 대면이름인 《DisplayObject》를 입력해가면 입력되는 대면이름에 부합되는 후보가 아래에 현시된다(그림 3-19).

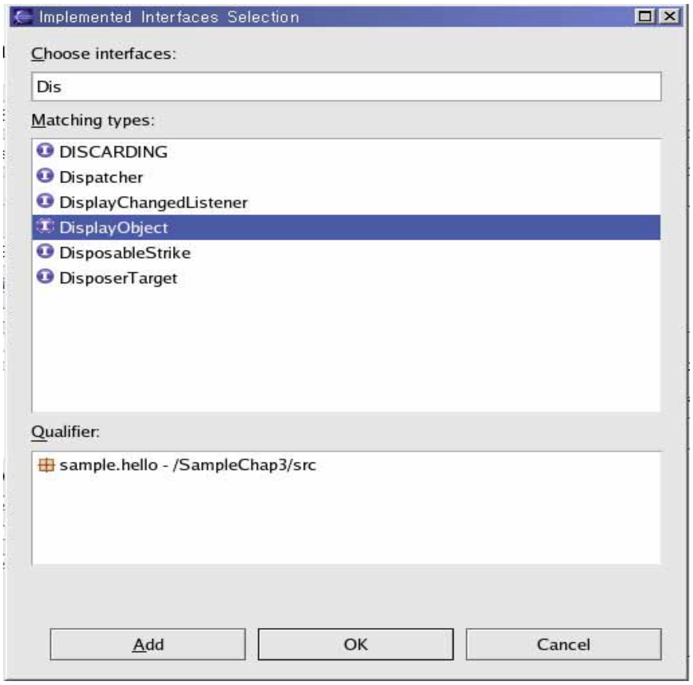


그림 3-19. 대면의 선택

DisplayObject를 선택하고 패키지가 이미 작성한 sample.hello로 되여있는가를 확인한 다음 [OK]단추를 찰칵한다. 계승된 추상메쏘드의 대용체(stub)를 작성하도록 선택표시칸이 표시되여있다는것을 확인한 다음 끝낸다(그림 3-20).

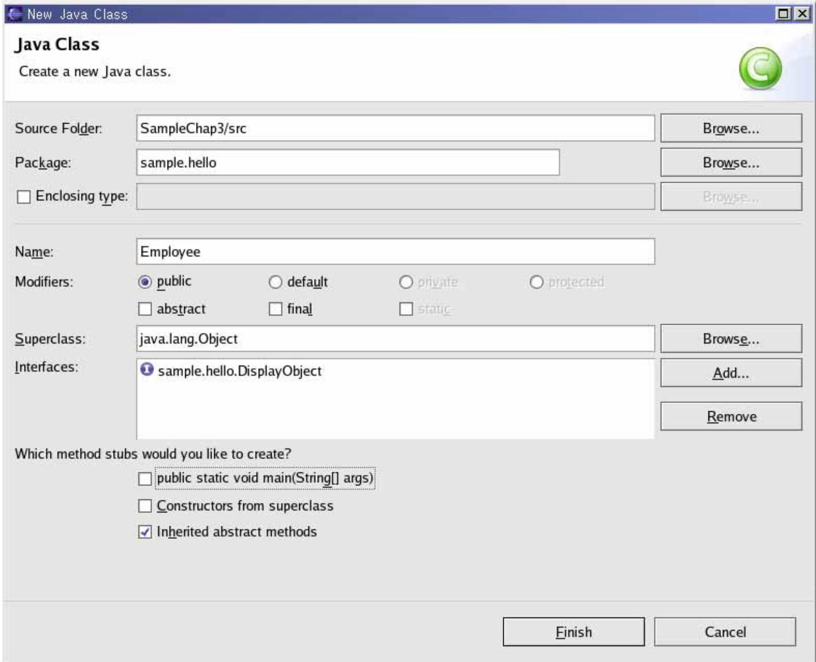


그림 3-20. 대면의 실현

3.2.7. Setter, Getter의 자동생성

JavaBean인 Employee.java에 Setter와 Getter를 추가한다. 우선 접근선언자 private의 속성과 구성자를 정의한다. 여기서는 종업원자료 즉 종업원번호(ID), 이름, 직위, 전화번호, 책임자에 대한 정보로 된 5가지 속성에 대하여 정의한다.

```
public class Employee implements DisplayObject {
    private String id;          // 종업원번호
    private String name;        // 이름
    private String role;              // 직위
    private String telNumber;         // 전화번호
    private String boss;              // 책임자

    public Employee(String id, String name, String role,String
                                    telNumber) {
      this.id = id;
      this.name = name;
      this.role = role;
      this.telNumber = telNumber;
    }
```

```
        public void display() {
        // TODO
        }
    }
```

다음으로 Java편집기상의 속성을 선언하고있는 부분에 유표를 놓고 지름차림표를 연 다음 [Source]→[Generate Getters and Setters]을 선택한다. 이때 나타나는 대화칸에서 생성하려는 메쏘드를 선택한다. 현재의 견본에서는 속성값을 설정할 때 구성자로 묶어서 진행하도록 하고있기때문에 여기서는 Getter만을 생성한다. [Select Getters]을 찰칵하면 매 속성의 get메쏘드가 선택표시된다. 이것을 확인한 다음 [OK]단추를 찰칵한다(그림 3-21).

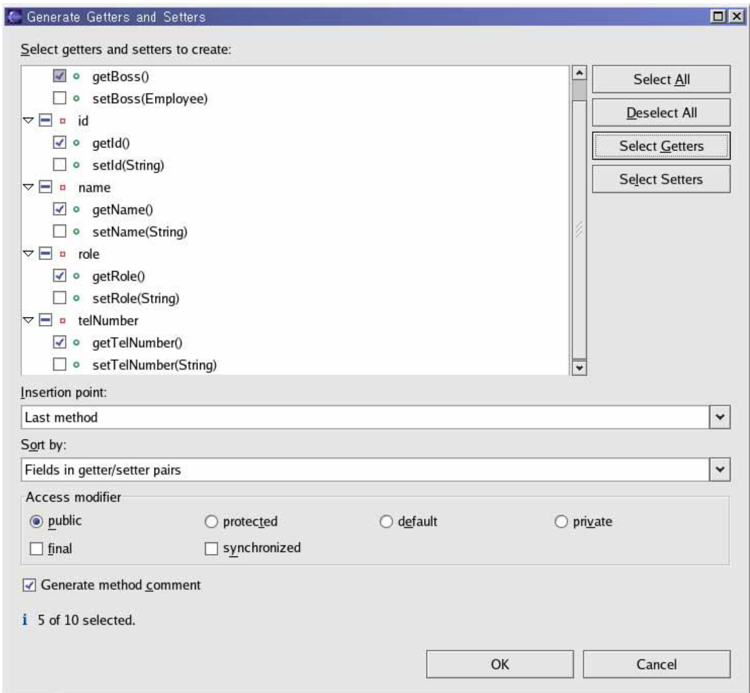


그림 3-21. Getter의 생성

Java편집기상에서 종업원번호(String id), 이름(String name), 직위(String role), 전화번호(String telNumber), 책임자정보(Employee boss)의 5개 속성에 대하여 생성된 Getter메쏘드를 확인할수 있다(그림 3-22).

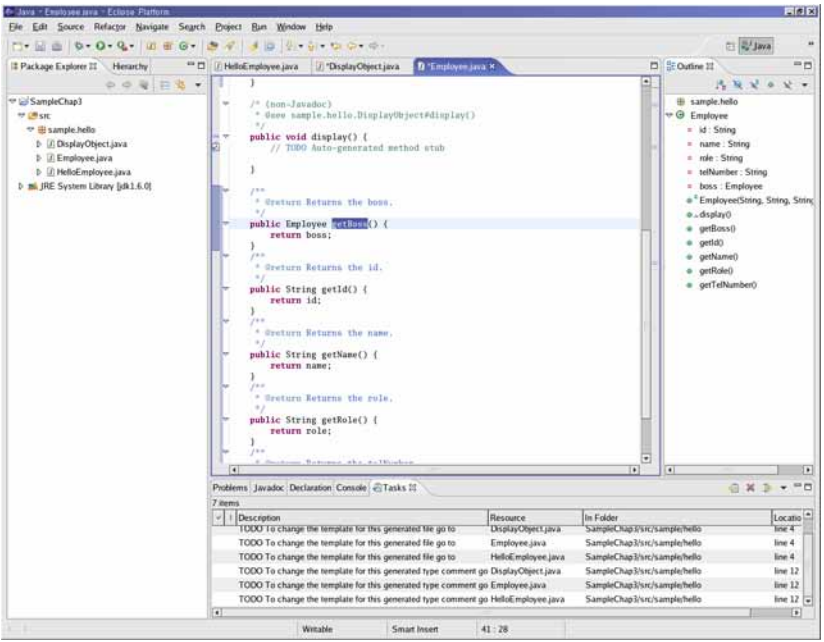


그림 3-22. Getter메쏘드가 추가된 다음의 Java편집기

Getter 및 Setter의 생성에서 판본2.1과 판본3.0의 차이점

판본3.0에서는 2.1에서의 다음과 같은 점이 변경되였다.

1. Javadoc해설문이 Getter용과 Setter용으로 각각 따로 지정할수 있게 되였다. 판본2.0까지는 Setter, Getter전용의 Javadoc해설문이 자동적으로 생성되였으나 국제화대응문제 등이 있어서 2.1에서 일부 기능이 삭제되였다. 그 대신 보통 새 메쏘드에 대하여 쓰이는 Javadoc해설문이 삽입되였는데 3.0에서는 각각 전용Javadoc해설문이 [Code Template]의 [Comment]설정에서 전용화할수 있게 되였다. ([코드형판]에 대하여서는 [3.2.3 보기 쉬운 원천코드의 작성]을 참고할것).

2. Setter의 인수가 마당이름에 의하여 만들어진다는 내용으로 다시 변경되였다. 판본2.1에서 Setter의 인수는 인수의 형에 기초하여 생성되도록 변경되였으나 판본3.0에서는 2.0까지에서와 마찬가지로 마당이름이 쓰이게 되였다.

```
String name;
public void setName(String string){ // 판본2.1
name = string;
}
        ↓
public void setName(String string){ // 판본2.0이전 및 3.0
this.name = name;
}
```

3.2.8. 반입기능

새로 Java원천코드를 작성할뿐아니라 반입(import)기능을 리용하여 다른곳에 있는 원천코드를 프로젝트에 추가할수 있다.

XML를 읽어들이기 위한 편의클라스(utility class)파일인 XmlUtil.java와 종업원속성을 넣은 EmployeeMap.java, EmployeeMap를 생성하기 위한 EmployeeMapFactory.java의 3개파일을 프로젝트의 sample.hello패키지에 반입한다.

우선 [Package Explorer]보기에서 반입목적지로 되는 sample.hello패키지를 선택한다. 다음 차림표띠의 [File]→[Import]를 선택하면 현시되는 대화칸에서 [File system]를 선택한다. [Next]단추를 찰칵하면 나타나는 대화칸의 [From directory]로 [root]/SampleProjects/Chapter3/SampleChap3
/src/sample/hello를 지정한다.(우의 지정경로에 파일이 있다고 가정한다.) Hello등록부에 포함되는 Java파일이 오른쪽에 표시된다.

여기서 반입대상들인

XmlUtil.java
EmployeeMap.java
EmployeeMapFactory.java

의 3개파일을 검사칸에서 선택한다. 반입목적지의 서류철 [Into folder]가《SampleChap3/src/sample/hello》로 되여있는가를 확인하고 [Finish]단추를 찰칵한다(그림 3-23).

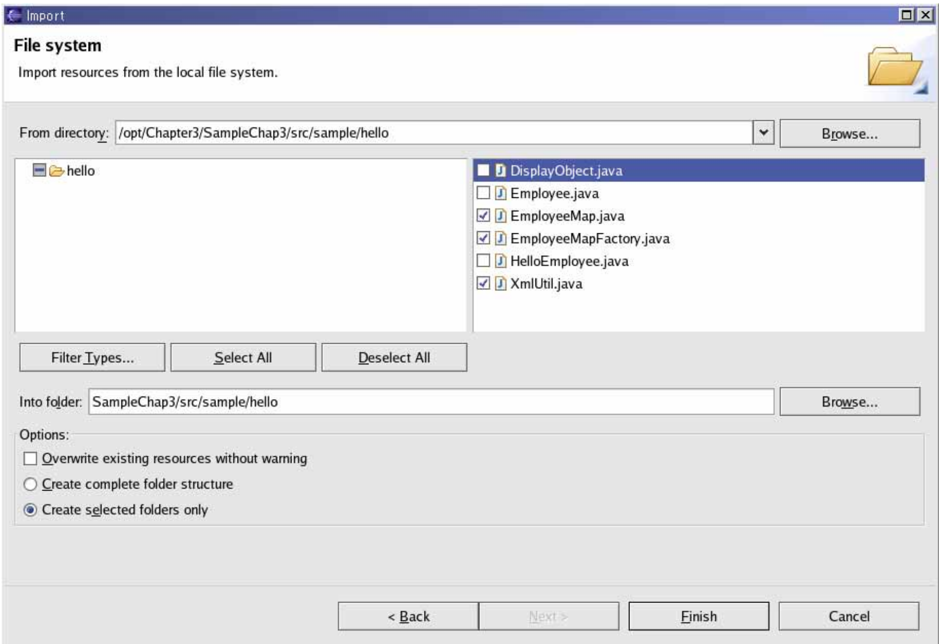


그림 3-23. 파일체계로부터의 반입

프로젝트 열기 — 프로젝트의 반입

현재 작업공간에 열려져있지 않은 Eclipse프로젝트를 작업공간상에 현시하는 기능도 Eclipse에서는 반입기능에 포함되여있다. 례를 들어 CD-ROM으로부터 견본프로젝트를 새로 Eclipse에 불러들여 참조편집하는 경우에 반입기능을 리용한다. 이러한 작업을 Visual Studio와 같은 다른 개발도구들에서는 《프로젝트 열기》라고 한다.

열려고하는 프로젝트를 《C/workspace/》에 복사한다. 복사한 등록부의 파일속성은 읽기전용으로 되여있는 경우가 있으므로 등록부이하에 있는 모든것들에 대하여 그 속성을 해제한다.

다음 Eclipse의 차림표띠에서 [File]→[Import]를 선택하면 현시되는 대화칸에서 [Existing Project into Workspace]를 선택한다. [Next]단추를 찰칵하고 [Project contents]의 [Browse]를 눌러 현시되는 대화칸에서 우에서 복사한 프로젝트의 등록부를 선택한다(그림 3-24). 프로젝트이름은 열려는 project파일에 이미 정의되여있기때문에 반입할 때 지정할수 없다.

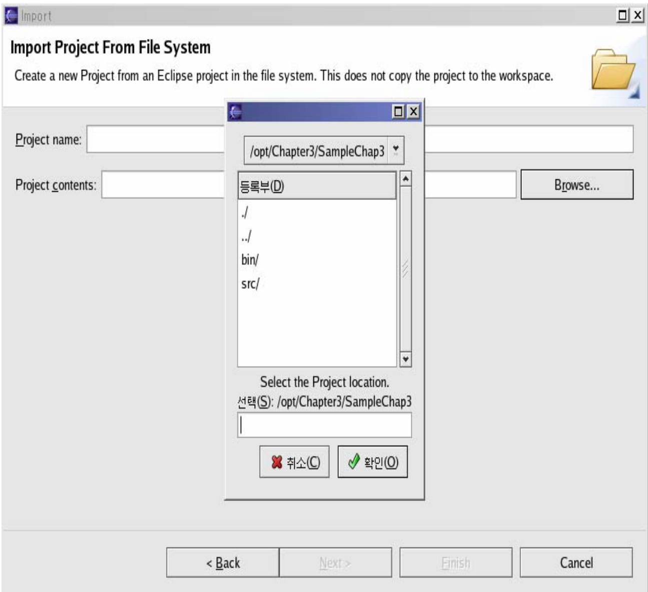


그림 3-24. 프로젝트의 등록부를 선택

3.2.9. 즉시수정기능

반입이 끝나면 [Package Explorer]보기에 콤파일오유를 가리키는 빨간 표식이 있을것이다. [Package Explorer]보기에서 오유가 있는 EmployeeMapFactor.java를 열면 Java편집기의 오른쪽끝의 회색부분에도 빨간 표식이 붙어있다. 여기를 찰칵하면 오유가 있는곳으로 유표가 이동된다. [Problems]보기에도 오유가 표시되기때문에 이것을 두번찰칵해도 오유가 있는곳으로 이동할수 있다(그림 3-25).

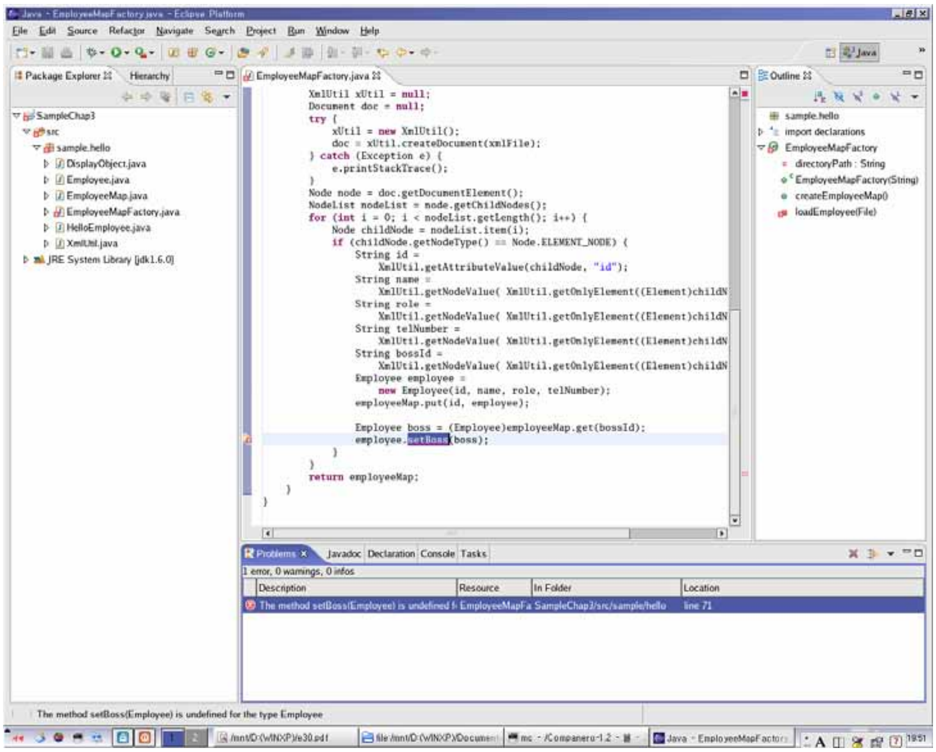
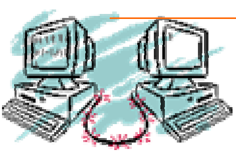


그림 3-25. 반입의 완료와 콤파일오유

오유행의 왼쪽끝부분에 노란색의 전구표식과 빨간색의 문제표식이 있는것을 확인할수 있다. 문제표식에 유표를 맞추면 오유내용이 표시된다. 또한 전구를 찰칵하면 즉시수정(quick fix)기능에 의하여 자동수정이 가능하게 되는 [Proposal Fix List]이 현시된다. 여기서는 수정후보로서

메쏘드 'setBoss(Eployee)'를 Employee.java에 작성한다.
형변환을 'employee'에 추가
파일안에서 이름변경한다.

의 3가지 내용이 현시된다(그림 3-26).



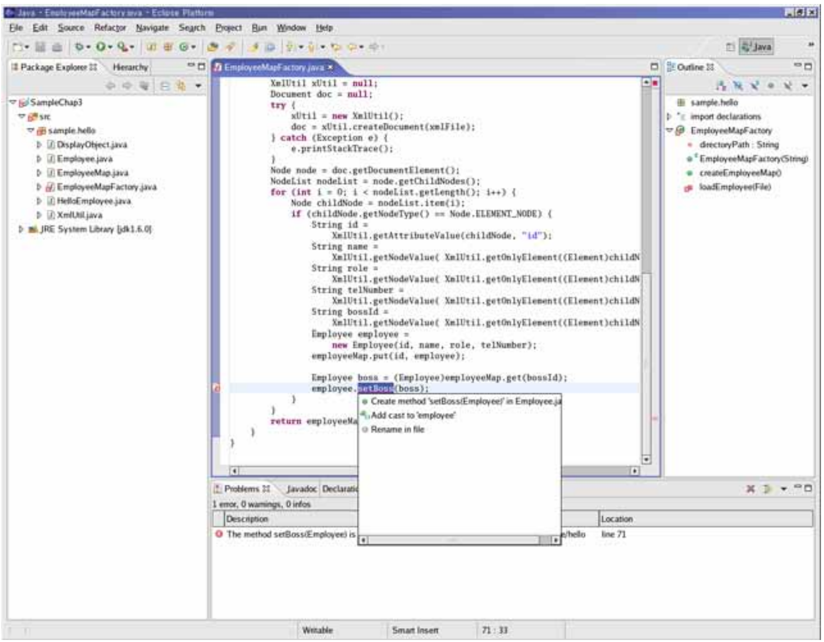


그림 3-26. 즉시수정기능

[3.2.7 Setter, Getter의 자동생성]에서 편집한 Employee.java에서는 Getter메쏘드만을 작성하고 Setter메쏘드는 작성하지 않았다. 오유가 생긴 EmployeeMapFactory.java에서는 속성 Boss에 대하여 Setter메쏘드를 설정해야 한다는것을 보여주고있다.

그래서 즉시수정기능을 리용하여 Employee.java에 setBoss메쏘드를 작성하기로 한다. 수정후보로 표시되여있는 [Create method 'setBoss(Employee)' in Employee.java(메쏘드 'setBoss(Employee)'를 Employee.Java에 작성한다)]를 선택하면 오른쪽에 도움말이 나타나고 삽입하려는 메쏘드가 표시된다. 이것을 확인한 다음 Enter건을 누르면 Employee.java에 대하여 setBoss메쏘드의 코드형판이 추가되므로 Setter메쏘드를 아래와 같이 입력한다.

```
public void setBoss(Employee boss) {
this.boss = boss;
}
```

여기서 Employee.java를 보관하면 [Problems]보기에 표시되여 있었던 오유도 없어진다(그림 3-27).

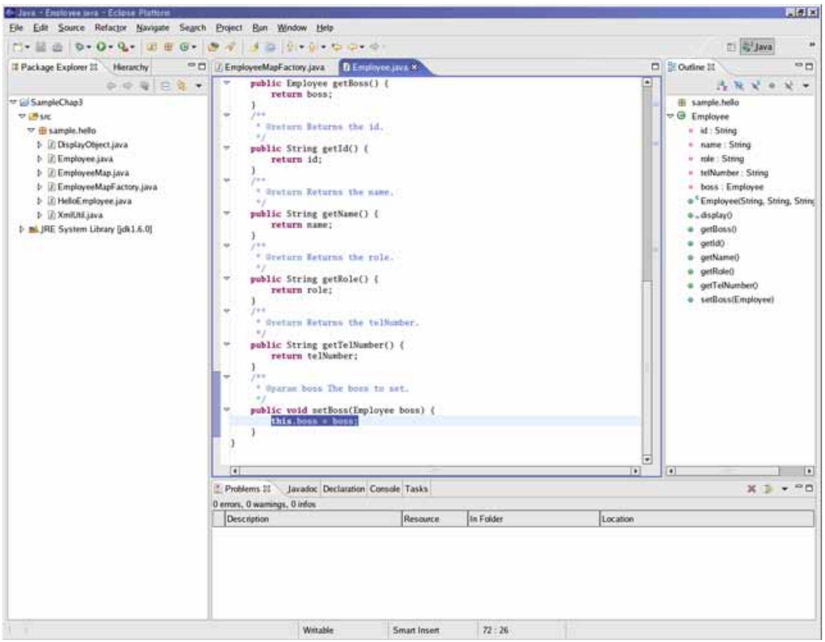


그림 3-27. setBoss의 추가

즉시수정기능을 리용하면 우에서와 같이 메쏘드를 추가하는것뿐아니라 부족한 import문과 입력한 철자법오유 등에 대하여서도 오유행 왼쪽끝의 노란색전구를 찰칵하여 수정후보를 선택하는 방법으로도 수정할수 있다. 지름건으로 Ctrl + 1이 할당되여있다.

오유수정은 아니지만 어느정도 원천코드를 써넣은 후에 국부변수의 이름을 변경하려고하는 경우에는 [Local name change]즉시수정기능을 리용하는 쪽이 편리하다. 변경하려는 변수에 유표를 맞추고 Ctrl + 1 또는 차림표띠에서 [Edit]→[Quick Fix]을 선택한다. 튀여나온 형판에서 [Rename in file]를 선택한 후에 변수이름을 바꾸어 주면 메쏘드안이나 Javadoc해설문안 등 다른 장소에서 쓰이고있는 변수의 이름도 동시에 변경된다.

3.2.10. 형판에 의한 자동기능

Eclipse에서는 [3.2.1. 메쏘드와 괄호의 자동설정]에서 본바와 같이 클라스에 포함되는 메쏘드나 마당의 자동설정기능을 가지고있을뿐아니라 사용자가 정의한 문자렬도 동일한 조작에 의하여 자동적으로 입력할수 있다.

일반적으로 자주 리용되는 문구들은 형판으로서 미리 등록되여있다.

try/catch문을 작성하는 실례를 보기로 하자.

Java편집기에서 try라고 입력하고 Ctrl + Space건을 누른다. 이것은 메쏘드를 자동설정할 때와 같은 조작이다. 이 조작에 의하여 내용물방조(contents assist)가 표시된다

(그림 3-28). 일반예약어인 [try]외에 [try-try catch block]라는 후보가 표시되는데 [try -try catch block]을 선택하고 Enter건을 누른다. Java편집기상에서 례외처리를 진행해야 하는 부분에 try/catch문이 작성되고 유표가 놓인다. Try문에서 발생하는 례외형을 입력하고 Tab건을 누르면 계속하여 catch문의 유효범위안으로 유표가 옮겨진다. catch문은 TODO과제로서도 추가된다.

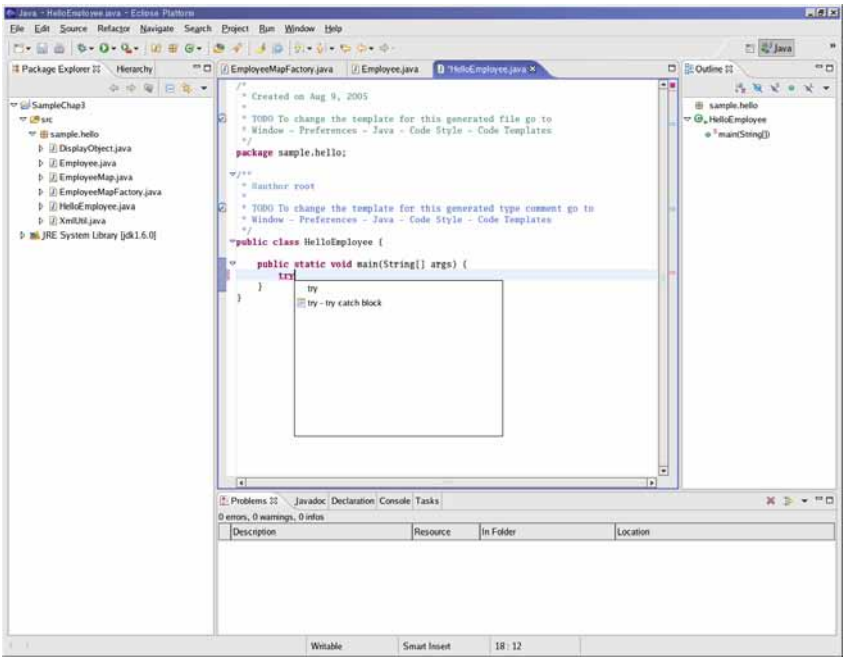


그림 3-28. 형판에 의한 자동삽입

그외에도

sysout	System.out.println
syserr	System.err.println
for	for문
while	while문
public_method	public메쏘드
main	main메쏘드

등 수많은 형판이 준비되여있다.

차림표띠의 [Window]→ [Preferences]을 선택하면 현시되는 대화칸에서 [Java] → [Editor]→[Templates]순서로 열어나가면 일람을 확인할수 있다. 이 대화칸에서 사용자가 독자적으로 형판을 새로 정의할수도 있고 이미 정의되여있는 형판을 편집할수도

있다. 그림 3-29안의 [Name]마당에 표시되여있는 문자를 입력하고 Ctrl + Space를 누르면 내용물방조가 현시된다.

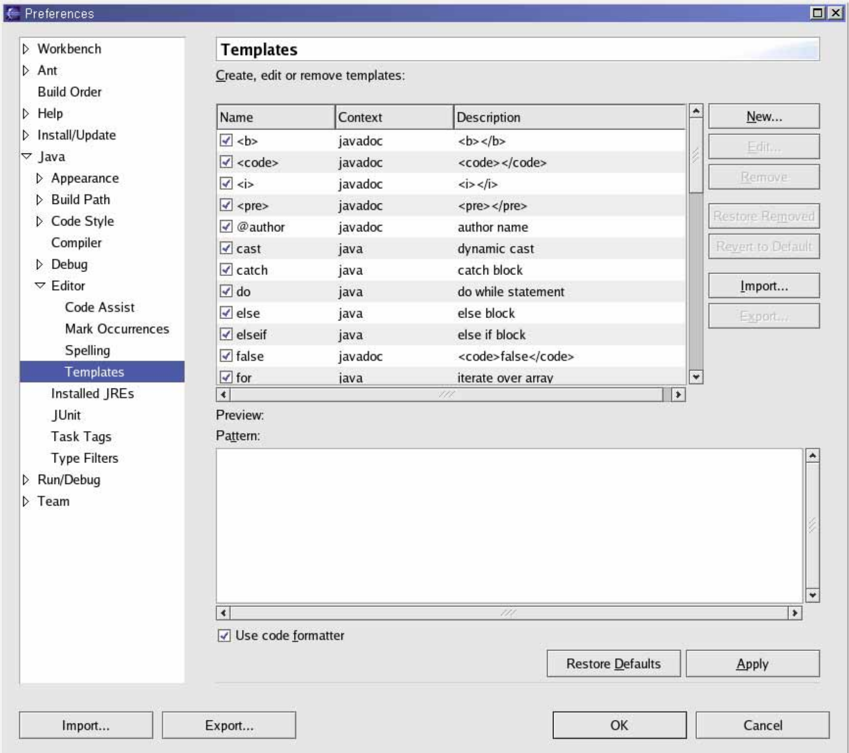


그림 3-29. 정의되여 있는 형판일람

3.2.11. TODO과제의 추가

현재 남아있는 TODO과제를 과제보기로부터 참조하면 Employee.java의 display 메쏘드가 아직 추가되여있지 않는것을 알수 있다.

```
public void display() {
// TODO
}
```

//TODO로 시작되는 행을 아래와 같이 변경함으로써 메쏘드를 삽입한다. TODO행을 삭제하면 과제보기의 목록도 삭제된다.

[3.2.10. 형판에 의한 자동기능]에서 소개한 방법을 리용해보기로 한다. sys라고 입력하고 지름건 Ctrl + Space로 내용물방조를 표시한다. sysout라는 이름으로 System.out.println()가 등록되여있으므로 이것을 선택한다.

```
/**
* 종업원정보를 표준으로 출력한다.
*/
public void display() {
    System.out.println("종업원번호=" + id);
    System.out.println("종업원이름=" + name);
    System.out.println("직위" + role);
    System.out.println("전화번호" + telNumber);
    System.out.println("책임자" + boss);
    System.out.println();
}
```

계속하여 Employee객체를 표시하기 위하여 아래의 메쏘드를 추가한다.

```
/**
* 이 객체를 문자렬화한다.
*/
public String toString() {
return "["
+ id     + ","
+ name + ","
+ role + ","
+ telNumber + ","
+ boss + "]";
}
```

3.2.12. 해설문, 해설문해제와 풍선도움말

어떤 부분의 원천코드를 한묶음으로 하여 해설문을 작성하려고 할 때에는 해당부분을 선택한 상태에서 지름차림표의 [Source]→[Toggle Comment]의 조작으로 행머리에 《//》를 삽입할수 있다. 이 조작에는 지름건이 할당되여있는데 원천코드를 선택한 후에 《Ctrl + /》를 누르면 동일한 동작이 가능하다. 해설문을 해제할 때에는 해당부분을 선택하고 다시 같은 건조작을 진행한다.

《/*》과 《*/》로 묶은 블로크해설문을 만드는 경우에는 해당부분을 선택한 상태에서 [Source]→[Add Block Comment]를 선택한다. 지름건은 《Ctrl +Shift + / 》이고 블로크해설문의 해제도 이 건을 리용한다.

여기서 HelloEmployee의 main메쏘드를 실례들어 본다. 먼저 main메쏘드의

《System.out.println("Hello World");》를 선택하고 《Ctrl + /》 로 해설문을 작성한다. 이 례는 한행이기때문에 그닥 편리성을 느끼지 못하지만 여러행에 대해서 해설문과 해설문해제를 반복할 때에는 매우 편리한 기능이다. 계속하여 EmployeeMapFactory클라스의 구체례를 XML파일을 인수로 하여 작성한다. createEmployeeMap메쏘드를 리용하여 종업원속성을 보존한 JavaBean인 Employee를 넣은 EmployeeMap를 작성한다. 마지막으로 EmployeeMap의 display메쏘드에 의하여 표준출력한다.

```
public static void main(String[] srgs) {
// System.out.println( "Hello World" );
        EmployeeMapFactory eFactory = new EmployeeMapFactory( "empl
   oyee.xml" );
        EmployeeMap eMap = eFactory.createEmployeeMap();
        eMap.display();
}
```

eMap.display()를 입력할 때 eMap.이라고 입력한 상태에서 잠간 기다리든지 아니면 Ctrl + Space를 누른다. 그러면 내용물방조에 입력가능한 객체의 성원(EmployeeMap에 포함되는 메쏘드)이 현시된다. 내용물방조의 항목을 선택하면 그 오른쪽의 풍선도움말에 설명문이 나타난다. 이 설명문은 원천코드안의 Javadoc해설문을 참고하고있다(그림 3-30).

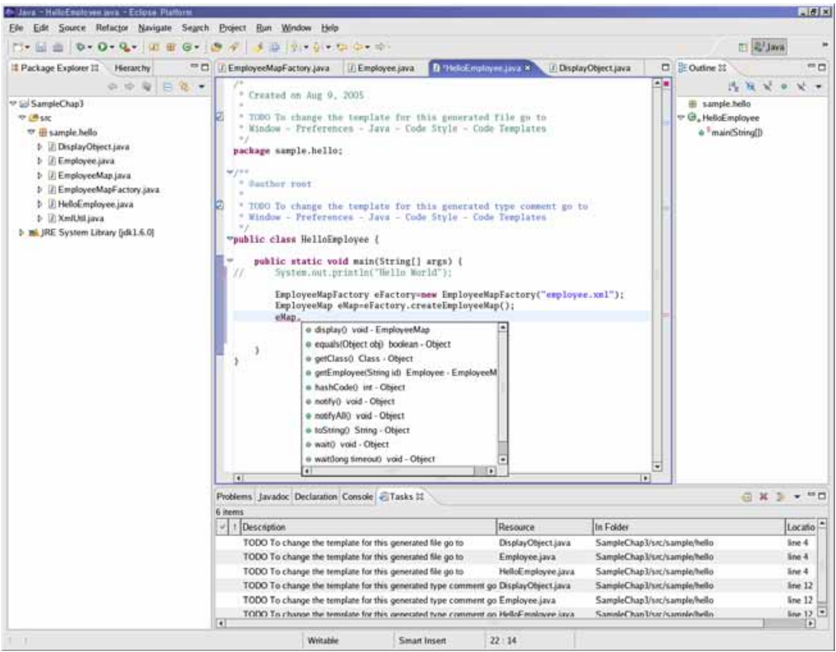


그림 3-30. Javadoc해설문을 리용한 참조

입력되여있는 메쏘드나 구체레에 유표를 맞추어도 풍선도움말에 표시되는 것과 동일한 설명문이 나타난다.

3.2.13. XML파일의 작성

견본프로그람이 읽어들이는 종업원정보가 들어있는 XML파일을 작성한다.

프로젝트서류철인 SampleChap3바로아래에 새로운 파일을 작성하기 위하여 [Package Explorer]보기에서 프로젝트 SampleChap3을 선택하고 지름차림표의 [New]→[File]을 선택한다. 또한 차림표띠에서 [File]→[New]→[File]을 선택하여도 같은 조작을 한것으로 된다. 생성하려는 서류철은《SampleChap3》으로 하고 파일이름은《employee.xml》로 한다(그림 3-31). [Finish]단추를 찰칵하면 빈 파일이 생성된다.

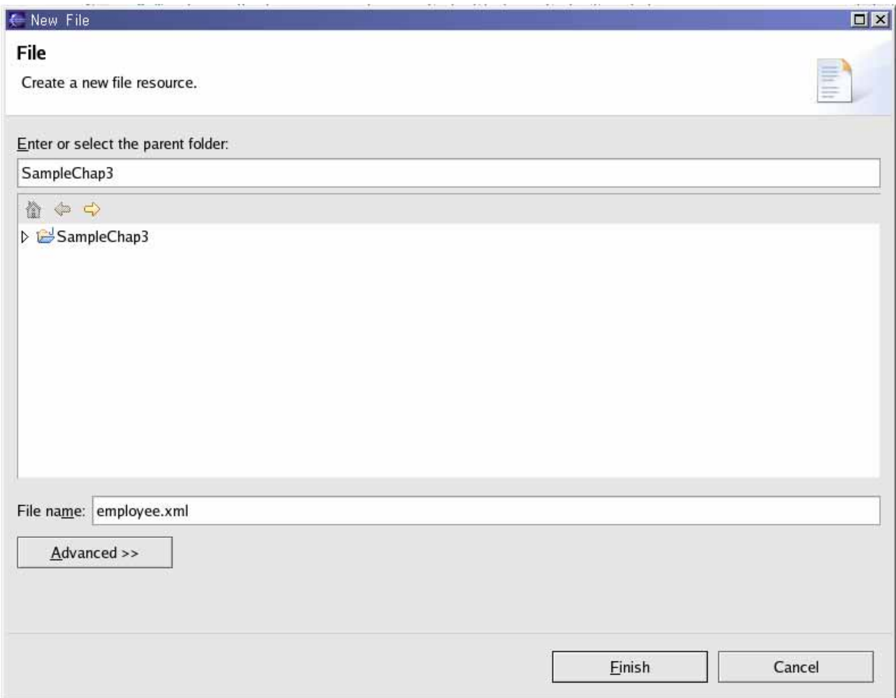


그림 3-31. 파일의 새로 작성

확장자와 리용하려는 편집기에 관한 설정은 차림표띠의 [Window]→[Preferences]을 선택하면 현시되는 대화칸의 [Workbench]→[File Associations]에서 설정할수 있다(그림 3-32).

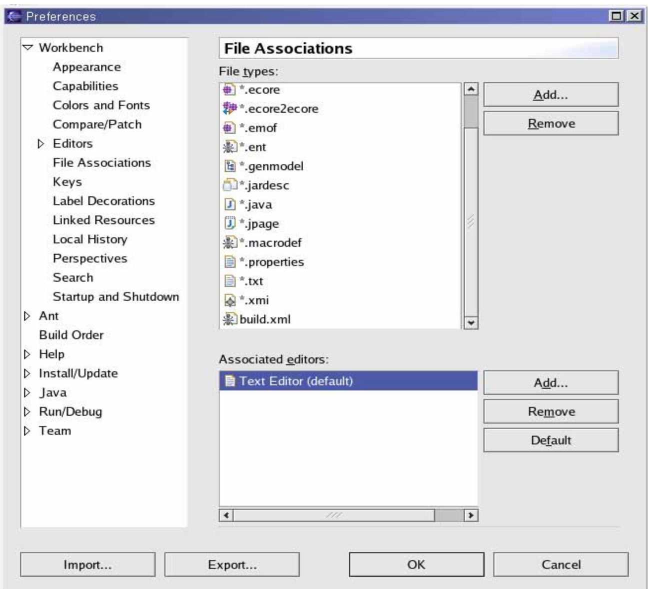


그림 3-32. 확장자와 파일형

[Add]단추를 찰칵하고 파일형에 《.xml》를 입력한다. 다음으로 [Associated editors]의 [Add]로부터 례하면 [Text Editor]를 선택한다. Eclipse에는 여러개의 XML편집기끼워넣기가 있으므로 요구하는 XML끼워넣기를 설치하여 사용할수 있다. Xml편집기를 리용하면 태그의 색갈이나 륜곽 등을 표시할수 있게 된다.

여기서는 목록 3-1에서 본 XML파일을 작성한다.

제3절. 실행과 기동구성의 설정

Java응용프로그람의 기동방법으로서 [3.2.2. 프로그람의 실행]에서 main메쏘드가 포함된 클라스를 선택한 다음 [Run]→[Run As]→[Java Application]을 선택하는 방법을 이미 소개하였다. HelloEmployee.java를 선택하고 이 방법으로 한번 실행해보면 Console보기에 XML로 서술한 내용이 출력된다.

이 절에서는 인수의 설정 등을 포함하는 보다 복잡한 실행방법과 그 관리기능에 대하여 설명한다. Eclipse에서는 프로그람을 실행하기 위한 설정을 기동구성이라고 부른다. 기동클라스 HelloEmployee.java에서는 종업원정보를 읽어들이기 위한 XML파일을 원천코드안에 서술하고있는데 이것을 프로그람인수로 지정하도록 변경한다. 먼저 프로그람을 아래와 같이 수정한다.

```
public class HelloEmploee {
  public static void main(String[] args) {

    EmployeeMapFactory eFactory = new
    EmployeeMapFactory( “employee.xml” );
    EmployeeMap eMap = eFactory.createEmployeeMap();
  eMap.display();
  }
}
```

— - — - — - — - — - — - — - — - — - — - — - — - — - — - — - — - — - — - —

```
public class HelloEmpyee {

  public static void main(String[] args) {
      if (args.length != 1) {
        System.out.println( “Usage: java sample.hello.
                      HelloEmployee XML_FILE);
        return;
      }
      EmployeeMapFactory eFactory = new
                      EmployeeMapFactory(args[0]);

    EmployeeMap eMap = eFactory.createEmployeeMap();
    eMap.display();
  }
}
```

프로그람을 수정한 후에 우에서 설명한 절차대로 실행하면 표준출력에 Usage가 현시된다.

그다음 인수를 지정하기 위한 기동구성을 새로 작성한다. 차림표띠에서 [Run]→[Run]을 선택하면 [Run]대화칸이 나타난다. 도구띠우의 실행단추의 내리펼침(pull-down)에서 [Run]을 선택하여도 같은 조작이 가능하다. 3.2.2에서 한번 실행한적이 있으면 《HelloEmployee》의 기동구성이 이미 Java응용의 아래에 추가되여있다.

이것과는 별도로 새로 기동구성을 작성한다. 대화칸왼쪽의 [Configurations]에서 [Java Application]을 선택하고 [New]단추를 찰칵하면 현시되는 대화칸에서 작성하려는 기동구성의 이름을 입력한다. 이름은 임의로 입력할수 있는데 여기서는 《SampleChap3-arg xmlfile》로 입력한다. [Project]에는《SampleChap3》이 기정값으로 입력되여있다. 계속하여 [Main class]를 지정한다. [Search]단추를 찰칵하면 프로젝트안의 클라스중에서 main메쏘드를 포함하는것이 검색된다(HelloEmployee.java를 선택한 상태에서 [Run]대화칸을 표시하고 새로작성을 진행하면 프로젝트, 주클라스의 입력마당들에 미리 기정값이 입력된 상태로 되기때문에 검색공정을 생략할수 있다). HelloEmployee를 선택하고 [OK]단추를 찰칵한다(그림 3-33).

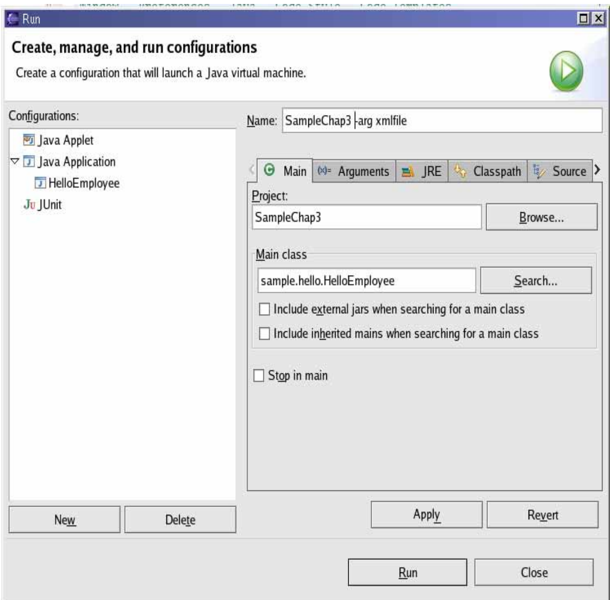


그림 3-33. 기동구성의 작성

계속하여 [Arguments]태브를 선택하고 Main메쏘드의 인수로 되는 [Program arguments]에 xml파일이름 《employee.xml》를 입력한다(그림 3-34). [Apply]단추를 찰칵하면 이 기동구성을 보존하고 실행한다.

이외에도 [Arguments]태브에서는 -D추가선택이나 -Xmx를 비롯한 가상기계인수와 작업등록부의 지정이 가능하다.

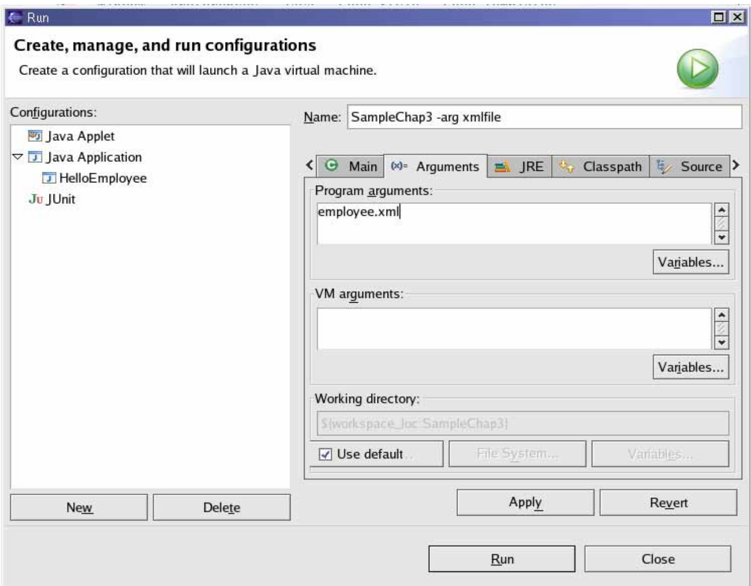


그림 3-34. 인수의 지정

JRE의 지정

Eclipse에서는 실행에 리용하는 Java실행환경의 판본을 기동구성별로 지정할수 있다. 일상적인 체계개발에서는 여러개의 JRE를 바꿔쓸 일이 많지 않지만 JRE판본의 차이에 의한 동작의 비교 등을 시험할 때에는 편리하다.

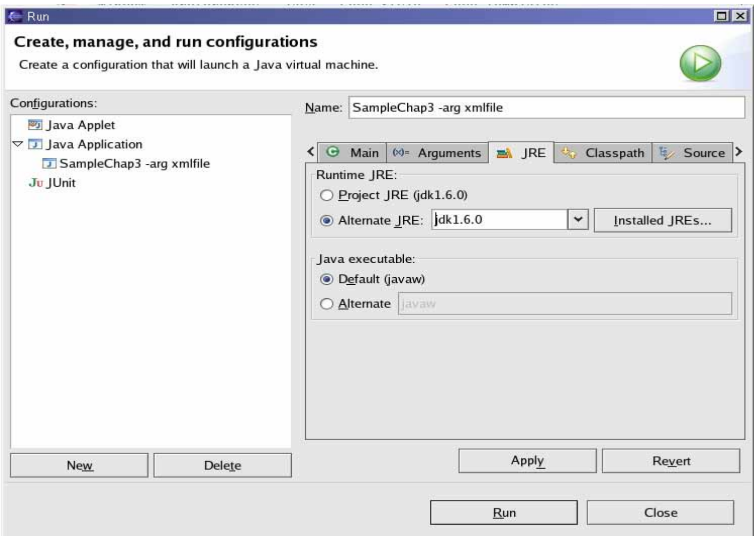


그림 3-35. JRE의 지정

JRE의 지정은 [Run]대화칸의 [JRE]태브에서 진행한다(그림 3-35).

차림표띠에서 [Window]→[Preferences]를 선택하면 현시되는 대화칸의 [Java]→[Installed JREs]에서 미리 등록해놓은 Java실행환경을 선택할수도 있고 웃그림의 대화칸에서 [Alternate JRE]를 선택하고 [Installed JREs]단추로 새로 Java실행환경을 추가할수도 있다.

제4장. Eclipse를 리용한 시험

제1절. 시험틀거리(testing framework)

최근에 쏘프트웨어부문에서 자주 framework라는 말을 쓰군 한다. 쏘프트웨어공학에서는 framework란 틀거리자체를 정해놓고 그 틀거리에 따라 목적을 실현한다.

이 장에서 소개하는 **시험틀거리**란 단위시험(unit test)을 자동화하기 위한 기틀 즉 단위시험을 자동화하기 위한 쏘프트웨어적인 틀거리이다. 체계개발에서 품이 가장 많이 드는 일중의 하나가 단위시험과 접수시험이다. 이러한 품을 덜어주는 시험기틀이라는 개념은 XP라는 개발방법론으로부터 생겨났다.

4.1.1. XP

XP는 eXtreme Programming의 략어로서 개발수법중의 하나이다. XP는 UP(Unified Process)와 같이 다량적인 문서를 요구하는 규모가 큰 무거운 방법론과는 달리 작성하는 문서의 량보다 코드의 질을 우선시하는 소규모의 가벼운 개발수법이다. XP를 실지로 적용하려고할 때에는 14가지의 준수사항을 지켜야 한다. 대표적인 준수사항에는 40시간로동, 개편, 쌍프로그람작성, 시험구동 등이 있다.

4.1.2. 시험구동프로그람작성

XP수법에 의한 프로그람개발에서 가장 큰 특징은 시험구동이다. 시험구동은 대상하는 프로그람을 작성하는것과 동시에 시험코드도 병행하여 서술한다.

시험구동으로 코드를 서술하는것이 가지는 우점은 다음과 같다.

- 외부명세서를 명확히 할수 있다.

외부명세서란 대상코드에 대한 내용을 말한다. 시험코드를 병행하여 서술함으로써 명세서에서 빠진것을 발견하거나 명세서가운데서 애매한 부분이 명백해지기도 한다. 그 결과 대상코드의 외부명세서가 명백해지게 된다. 삽입되는 코드의 외부명세서가 명백해지면 코드의 수정 및 확장이 쉬워진다.

- 코드의 품질이 향상된다.

시험과 병행하여 코드가 서술된다는것은 결국 서술된 코드는 항상 시험을 통과한 상태에 있게 된다는것을 의미한다. 또한 시험에서 통과되려면 정확한 코드를 써야 하기때문에 서술한 코드의 품질은 분명히 높아지게 된다.

- 회귀시험을 자동화할수 있다.

회귀시험은 프로그람을 수정할 때 다른 개소에 영향이 파급되여있지 않는가를 확인하는 시험이다.

처음에 서술한 시험코드는 회귀시험을 진행할 때 재리용할수 있다. 시험틀거리는 서술한 코드의 실행을 자동화하는 물림새이므로 필연적으로 회귀시험도 자동화되게 된다.

- 개편을 진행한다.

개편을 진행하면 외부로부터 본 프로그람의 거동을 변화시키지 않고 프로그람을 깨끗하게 수정할수 있다. 그러나 동작하고있는 프로그람을 변경하기때문에 실지로 거동에서 변화가 없는지 의문을 가질수 있다. 거동이 변했는가를 확인하기 위하여 회귀(regression)시험을 진행한다. 시험을 통과하면 거동이 변하지 않았다는것이 담보된다.

대상코드와 시험코드를 병행하여 서술하면 대상코드의 외부명세서가 명백해진다. 그러한 의미에서 시험코드는 정확한 외부명세서라고 볼수 있다. 왜냐하면 시험코드는 프로그람코드로서 서술되며 시험에 통과되였다면 시험에서 서술된 명세서대로 코드가 동작한다는것이 담보되기때문이다.

물론 보수나 수정에서는 자연언어로 서술된 명세서가 필요하다. 그러나 자연언어로 쓴 명세서는 실행을 할수 없지만 시험코드는 실행가능하다. 그러므로 자연언어로 서술된 명세서에 비하면 압도적으로 프로그람의 검증이 헐해진다.

시험과 병행하여 코드를 작성한다.

론리가 복잡한 코드를 작성하는 경우에는 시험과 병행하는 방법이 목표가 뚜렷해질수 있다. 한편 목표가 제대로 정해지지 않은 경우에는 서술하는 코드의 역할도 명확하지 않을 때가 많다. 복잡한 코드를 작성할 때 시험과 병행하여 코드를 서술하는 쪽이 실지로 완성할 때까지의 시간이 단축되군한다. 그러므로 복잡한 코드를 쓸 때에는 시험을 병행하여 코드를 작성하는것이 좋다.

4.1.3. 단위시험, 접수시험

XP수법에서 시험은 단위시험과 접수시험(acceptance test)의 2가지로 구분된다. **단위시험**은 단일체시험이라고도 부르는데 주로 쏘프트웨어개발자의 립장에서 진행하는 시험이며 일반적으로는 시험구동식프로그람개발에서 작성되는 시험코드를 가리킨다. 한편 **접수시험**은 쏘프트웨어리용자의 립장에서 진행되는 시험이며 사용자가 실지로 그 쏘프트웨어를 리용하는 경우에 시험이 실시된다. 이 두 시험 가운데서 단위시험을 지원하고 회귀시험을 자동화해주는 도구가 JUnit이다.

제2절. JUnit의 설정

여기서는 JUnit를 사용한 시험구동프로그람작성에 대하여 취급한다. JUnit를 쓰려면 《http://www.junit.org》로부터 코드를 내리적재하거나 Eclipse의 plugins등록부에 있는 JUnit3.8.1을 리용한다. JUnit의 건립경로설정, 클라스경로변수 JUNIT_HOME 등을 설정하는 구체적인 방법에 대하여서는 제3장을 참고하면 된다.

제3절. JUnit의 기초

4.3.1. 시험구동프로그람작성

먼저 간단한 례를 들어 JUnit의 사용방법에 대하여 설명하기로 한다. 앞장에서도 나온 Hello World의 실례를 가지고 JUnit의 시험구동프로그람을 작성한다. 이제 다음과 같은 경우를 가정해본다.

어떤 프로젝트에서 여러가지 클라스에 인사를 보내는 Hello라는 클라스를 만든다. 이것은 프로젝트안에서 의사소통을 잘 하기 위한 클라스이다.

그러면 이 클라스의 명세서는 다음과 같이 구성할수 있다.

Hello클라스는 String getGreeting(String name)라는 메쏘드를 가진다. 이 getGreeting메쏘드는 인수(name)의 문자렬앞에 문자렬《Hello》를 붙이고 되돌이값을 돌린다.

개발자들은 일반적으로 명세서가 결정되면 메쏘드의 작성을 인차 하고싶어한다. 사실 지금 만들려는 메쏘드는 명세서가 단순하기때문에 메쏘드본체를 직접 쓰기 시작한다고 해도 지장은 없을것이다. 그러나 여기서는 시험을 병행하여 메쏘드본체를 만들어 보기로 한다.

그런데 빈 getGreeting메쏘드만 써넣으면 Eclipse의 Task보기에 《String형의 결과를 되돌릴 필요가 있다.》라는 경고통보가 표시된다. 이것을 피하기 위하여 림시로 메쏘드에 《return null;》라고 써놓는다.

목록 4-1. 처음에 고의적으로 실패하는 례(Hello.java)

```
package sample.hello;
public class Hello {
   public String getGreeting( String name ) {
   return null;
   }
}
```

시험구동프로그람을 작성하기로 한다. 우선 [Package Explorer]에 있는 sample.hello패키지를 선택한 다음 지름차림표를 연다. 다음 [New]→[JUnit Test Case]를 선택하여 [New JUnit Test Case]대화칸을 연다. [Name]본문칸에 《HelloTest》라고 입력하고 [Which method stubs would you like to create?]의 [constructor()]를 선택한다. 다음 [Finish]단추를 찰칵한다(그림 4-1).

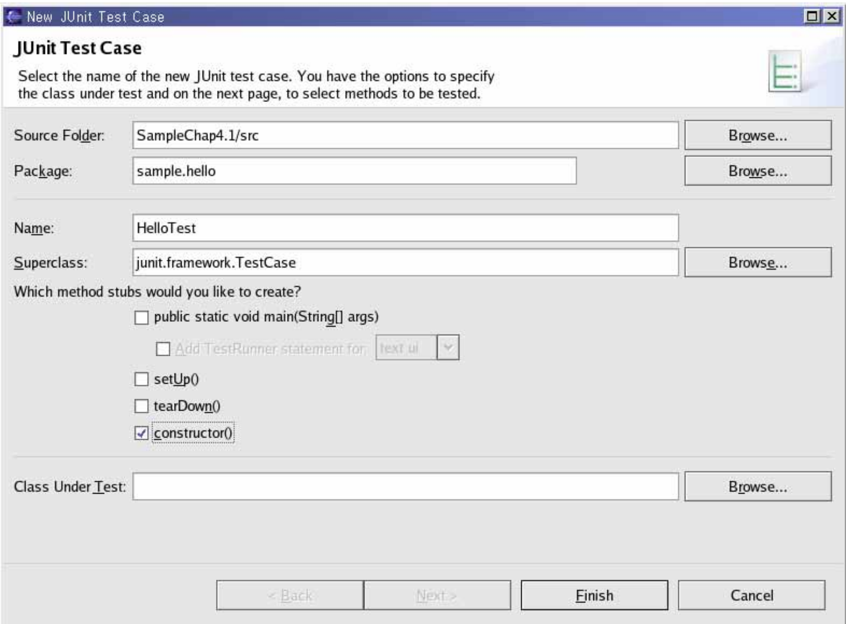


그림 4-1. 시험코드의 작성

목록 4-2가 시험코드이다. 시험경우인 HelloTest클라스는 TestCase클라스를 계승한다. 또한 구성자는 String형의 인수를 하나 취하고 상위클라스(super class)의 구성자에 넘기고있다. 이 구성자는 시험경우작성의 조수를 리용하면 자동작성된다.

시험경우인 HelloTest클라스에 메소드 《public void testGetGreeting() throws

Exception》를 추가한다. 여기서 메쏘드의 앞머리에 test라는 문자렬을 추가해야 한다는데 주의해야 한다. 이것을 추가하지 않으면 시험경우 실행할 때 시험경우로 인식되지 않는다. 또한 시험경우 메쏘드의 되돌이값은 void로 한다. 시험경우를 서술할 때 되돌이값이 없다고 생각해도 무방하다. 마지막으로 throws Exception은 례외가 일어난 경우에 Eclipse의 [JUnit]보기에 례외정보를 현시하기 위해서이다. TestCase클라스에는 시험용메쏘드가 여러개 준비되여있다. 그중에서 가장 많이 쓰는것이 assertEquals메쏘드이다.

이 메쏘드의 형식은 다음과 같다.(인수의 순서는 바꿀수 없다.)

assertEquals(구하려는 결과, 실지로 메쏘드를 호출한 결과)

목록 4-2코드에서도 assertEquals(구하려는 결과:String, 메쏘드호출:String)의 형식을 리용하고있다. Hello클라스의 getGreeting메쏘드의 명세서는 앞에서와 같으므로 구하려는 결과는 《Hello World》라는 문자렬로 된다.

JUnit가 제공하는 메쏘드는 표 4-1과 같다. 이 장에서는 실지 시험경우에 자주 사용하는 assertEquals메쏘드를 중심에 놓고 해설한다.

목록 4-2. 시험코드(HelloTest.java)

```
package sample.hello;
import sample.hello.Hello;
import junit.framework.TestCase;

public class HelloTest extends TestCase {
// 이 구성자는 JUnit의 시험경우작성조수에 의하여 자동생성된다
   /**
    * Constructor for HelloTest.
    * @param arg0
    */
   public HelloTest(String arg0) {
      super(arg0);
   }
   // TestCase① : 메쏘드이름의 앞머리에는 반드시 “test”를 단다
   public void testGetGreeting() throws Exception {
      Hello hello = new Hello();
      /*
       * Hello클라스의 String getGreeting( String literal)은
       * “Hello”문자렬을 인수의 문자렬앞에 붙여서 되돌리는 메쏘드
       */
      assertEquals( “Hello World”, hello.getGreeting( “World”));
   }
}
```

시험경우 클라스의 이름과 관리방법

시험경우의 클라스이름짓기규칙에 대해서는 일정한 론의가 있다. 례하면 앞의 실례에서는《HelloTest》라는 이름을 시험경우에 붙이고있다. 그러나《TestHello》라는 이름을 붙이는것이 더 좋다고 하는 사람도 있다.

이름을 다는 문제는 개발자에 의해 결정되는 문제이지만 시험경우 클라스를 넣는 등록부를 프로젝트안에 만들고 시험경우만을 넣는 경우에는 머리에 Test를 붙이는것보다 꼬리에 Test를 붙이는쪽이 혼돈하지 않아서 좋을것이다. 반대로 시험경우와 시험대상클라스를 프로젝트안의 같은 등록부에 넣는 경우에는 머리에 Test를 붙이는 쪽이 시험경우이라는것을 알아보기 쉬워질수 있다. 단지 이름의 첫머리에 T가 붙은 클라스가 있는 경우에는 그 클라스가 시험경우클라스와 혼돈되기 쉽다.

일반적으로 원천을 src등록부에, 시험경우는 test등록부에 넣고 따로따로 관리하는 방법이 있다. 이렇게 하면 프로젝트안에서 서로 다른 등록부에 의하여 관리되고있어도 같은 패키지로 취급할수 있으므로 패키지내유효범위의 메쏘드도 시험이 가능하게 된다.

프로젝트안의 서로 다른 등록부에서 시험대상코드와 시험코드를 갈라내는 방법에 대해서는 앞으로 설명한다.

4.3.2. 고의적으로 실패시키는 경우

목록 4-2와 같이 시험코드를 작성한 다음 [Package Explorer]안의 HelloTest.Java를 선택한 상태에서 지름차림표를 열고 [Run]→[1 Junit Test]을 선택한다. 그러면 시험경우가 실행된다. 목록 4-2와 같이 원천코드를 작성한 경우 메쏘드의 삽입이 불완전하므로 시험은 분명히 실패한다(그림 4-2). 만일 시험에서 성공했다면 다른 시험이 진행될 가능성이 있다.

처음에 고의적으로 실패시키는 리유는 자기가 진행하고있는 시험경우의 클라스가 목적하는 경우인가를 확인하기 위해서이다. 실패해야 할 시험이 성공하거나 원래의 의도와는 다른 오유통보가 출력될 때에는 거의나 다른 시험을 실행하였기때문에 그러한 결과가 현시된것이다.

현시된 [JUnit]보기의 [Failure Trace]을 보면 [expected:]에는《Hello World》라고 표시되고 [but was:]에는《null》이라고 표시되여있다(목록 4-3). 즉 Hello World라는 문자렬을 결과로 기대했는데도 실지 결과는 null이였기때문에 시험이 실패했다는것을 보여주고있다. 이러한 실패를 JUnit에서는 고장(Failure)이라고 부른다. JUnit에서 시험의 실패는 고장과 오유(Error)의 2가지로 갈라본다. 고장은 예견한 결과와 실지로 얻은 결과가 다른 경우에 일어난다. 한편 오유는 예견하지 않은 례외나 오유가 시험경우의 실행중에 발생한 경우에 일어난다.



그림 4-2. 시험의 실패

목록 4-3. 실패

```
junit.framework.ComparisonFailure: expected:<Hello World> but was:<null>
```

4.3.3. 고의적으로 성공시키는 경우

다음 단계에서는 고의적으로 시험을 성공시킨다. 결과 다음과 같은 효과를 얻을수 있다.

1. 코드에 대한 믿음감을 가질수 있다.
2. 조금씩 메쏘드의 삽입을 진행할수 있다.
3. 오유(bug)의 발생이 적어지며 순조롭게 진행할수 있다.

우에서 렬거한 1.~3.의 효과는 서로 밀접히 련관되여있다. 시험구동프로그람의 개발방법에서는 조금씩 삽입하고 조금씩 시험하는 방식을 취한다. 고의적으로 성공시키는 방법이 바로 메쏘드를 증분식(incremental)으로 삽입해가는 첫 공정으로 된다. 또한 조금씩 시험과 삽입을 반복함으로써 삽입시 오유의 발생원인이 적어지고 순조롭게 코드작성을 진척시킬수 있다. 또한 시험을 실행하는 과정에 푸른띠가 나와있는 한 자기가 코드를 작성해가는 방향이 틀리지 않는다는것을 확인할수 있으므로 침착하게 코드작성을 진행할수 있다.

목록 4-4가 고의적으로 성공시키는 경우에 대한 실례이다. 시험경우는 Hello World라는 결과를 기대하고있다. 목록 4-4에서는 고의적으로 성공시키기 위하여 《Hello World》라는 문자렬만을 되돌리는 메쏘드를 삽입하고있다. 이제 다시 시험경우

를 실행해본다. 그림 4-2에 있는 전번시험의 재실행단추를 찰칵하면 간단히 시험을 재실행할수 있다. 시험경우를 실행하면 그림 4-3과 같이 푸른띠가 나오고 시험은 성공한다. 실패하는(빨간띠가 나오는) 경우에는 다른 시험경우를 실행하고있지 않는가를 검사해본다.

Eclipse3.0부터는 《Failure Trace: "junit.framework.ComparisonFailure: expected...》를 두번찰칵하면 기대한 값과 결과를 비교할수 있는 [Result Compare]창문이 열리게 되였으므로 이러한 경우에 리용하면 편리할것이다.

목록 4-4. 고의적으로 성공시키는 실례(Hello.java)

```
package sample.hello;

public class Hello {
//고의적으로 성공시키기 위하여 "Hello World"
//라는 문자렬만 되돌리는 메쏘드를 작성한다.
public String getGreeting( String name ) {
        return "Hello World";
 }
}
```



그림 4-3. 고의적으로 성공시킨 경우

4.3.4. 메쏘드의 삽입과 재시험

《Hello World》라는 결과만을 되돌리는 메쏘드는 실지 사용할만한 메쏘드가 못된다. 그래서 실지 사용할만한 실용적인 메쏘드를 삽입한다. 이getGreeting메쏘드의 명세서는《인수로 문자렬을 받고 그 문자렬앞에 Hello라는 문자렬을 붙인다.》는 내용을 담고있다. 코드를 목록 4-5에 제시한다.

목록 4-5. 실용적인 메쏘드의 삽입(Hello.java)

```
package sample.hello;
public class Hello {
  public String getGreeting( String name ) {
  return “Hello” + name;
  }
}
```

시험을 실행시켜보면 성공할것이며 실패할 때에는 오유원인을 찾아보아야 한다. 그러나 성공해도 고의로 성공시키고있기때문에 다른 경우 (“Hello World”이외의 경우)에도 성공되는가가 문제로 된다.

그래서 시험경우를 바꿔써도 삽입할수 있는가를 검증하기로 한다. 수정한 시험경우를 목록 4-6에 제시한다. 여기서 주의를 돌려야 할것이 해설문에서 《2단계시험》이라고 쓴 부분이다. 먼저번 시험에서는 《Hello World》라는 문자렬의 검사만을 하였다. 이번에는 하나 더 《Hello Java》라는 문자렬에 대응가능한가를 검사하고있다. 이 2개의 례로 메쏘드가 정상적으로 삽입되였는가를 기본적으로 검사할수 있다. 이처럼 현시할 내용이 하나인 경우(《Hello World》)만이 아니라 여러개인 경우(《Hello World》와 《Hello Java》)를 써서 시험경우메쏘드를 서술하는 방법을 삼각측량(Triangulation) 또는 여러단계시험이라고 한다.

목록 4-6. 시험경우의 수정 (HelloTest.java)

```
public void testGetGreeting() throws Exception {
Hello hello = new Hello();
// 1단계시험
assertEquals( “Hello World”, hello.getGreeting( “World” ) );
    // 2단계시험
    assertEquals( “Hello Java”, hello.getGreeting( “Java” ) );
}
```

4.3.5. 시험경우와 삽입메쏘드의 약간한 수정

삽입코드를 보면 getGreeting메쏘드가 Hello클라스의 내부상태에 의존하지 않는다는것을 알수 있다. 즉 이 메쏘드의 사용자는 getGreeting메쏘드를 사용할 때 일일히 Hello클라스의 구체례를 생성할 필요가 없다. 그러므로 삽입메쏘드에 static열쇠어를 붙여서 클라스메쏘드화한다(목록 4-7). 그 다음 재시험을 진행해본다.

목록 4-7. 메쏘드의 수정 (Hello.java)

```
package sample.hello;
public class Hello {
  // 클라스메쏘드로 변경하였다.
  public static String getGreeting( String name ) {
  return “Hello” + name;
  }
}
```

일단 시험은 성공하지만 시험코드가 Hello클라스의 구체례를 생성하고 그 구체례로부터 static메쏘드를 호출하고있는것은 문제가 있다(Eclipse의 콤파일러의 기정설정에서는 경고가 나오게 되여있다). 왜냐하면 구체례로부터 static메쏘드를 호출하는것자체에는 오유가 없으나 후에 이 코드를 읽는 사람이 이 메쏘드를 구체례메쏘드로 잘못 리해할 가능성이 있기때문이다. 즉 구체례의 내부상태에 의존하는 메쏘드라고 생각할수 있다. 이러한 일이 없도록 하자면 코드삽입자가 클라스메쏘드를 사용하고 있다는것을 명시해놓을 필요가 있다. 개선한 시험경우의 코드를 목록 4-8에 제시한다.

목록 4-8. 개선된 시험경우(HelloTest.java)

```
// getGreeting메쏘드를 클라스메쏘드로서 호출하도록 변경
public void testGetGreeting() throws Exception {
  // 1단계시험
  assertEquals( “Hello World”, Hello.getGreeting( “World” ) );
  // 2단계시험
  assertEquals( “Hello Java”, Hello.getGreeting( “Java” ) );
}
```

4.3.6. 시험구동에 의한 메쏘드삽입순환

그림 4-4에서 시험구동에 의한 메쏘드삽입순환을 보여주었다. 이 그림에서 특히 중요한것은 [대상메쏘드의 수정]→[시험경우의 수정]의 순환이다. 이 순환은 작성자가 메쏘드의 코드를 완전히 작성할 때까지 계속한다. 메쏘드의 수정에서는 제7장에서 소개하는 개편의 수법을 적용하면 순조롭게 수정할수 있다.

이 그림에서는 시험경우의 서술을 먼저 써놓았지만 경우에 따라서는 삽입메쏘드를 먼저 쓴 다음에 시험메쏘드를 서술해도 무방하다. 여기서 중요한것은 삽입메쏘드와 시험메쏘드 쌍방의 서술을 병행하여 진행하는것이다. 특히 론리가 복잡하여 비교적 삽입이 힘든 메쏘드에서는 그림과 같은 순환을 리용하는것이 매우 좋다. 물론 지금 취급하고있는것처럼 간단한 메쏘드의 경우에는 처음부터 메쏘드를 삽입해도 된다.

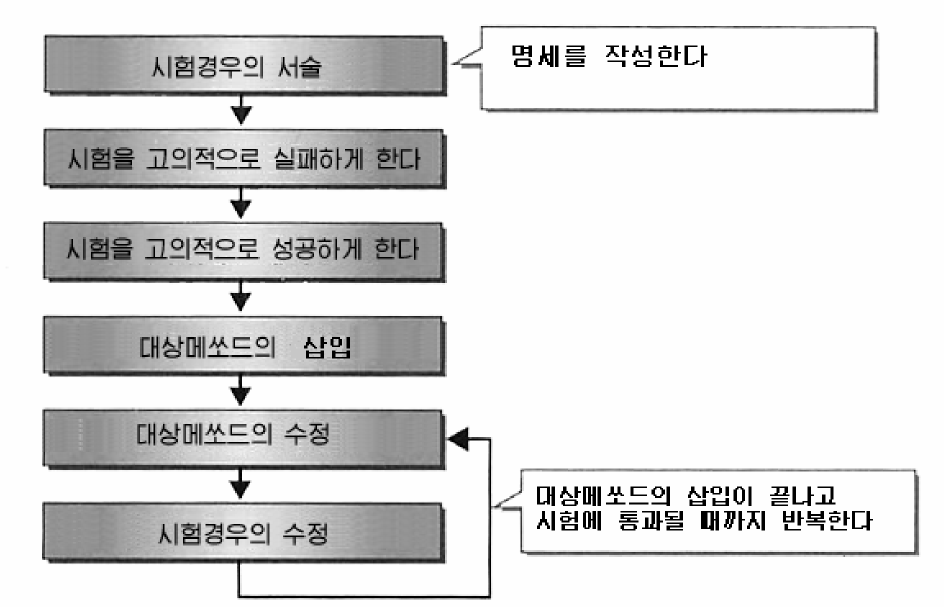


그림 4-4. 시험구동에 의한 메쏘드삽입순환

제4절. 시험경우의 본격적인 작성

여기서는 보다 실용적인 례를 들어 시험경우를 서술한다.

4.4.1. 원천서류철의 생성

처음에 취급한 《Hello World》의 례에서는 시험경우의 클라스와 원천코드의 클라스를 같은 등록부의 같은 패키지에 넣었다. 그러나 실지개발에서는 시험경우의 등록부와 원천코드의 등록부를 갈라써야 할 때가 있다. 그렇게 하기 위한 절차는 다음과 같다.

SampleChap4.1프로젝트를 선택한 상태에서 지름차림표를 열고[New]→[Source Folder]을 선택한다. 그리고 《test》라는 이름으로 등록부를 생성한다(그림 4-5). 여기서 문제로 되는것은 메쏘드의 유효범위가 패키지유효범위인 경우 같은 패키지에 시험클라스가 없으면 시험이 불가능하게 된다는것이다. 그러나 이제부터 설명하는 절차를 따르면 문제가 없다. src등록부에 있는 패키지(sample.hello패키지)와 같은 이름의 패키지를 test등록부에 추가한다(구체적인 방법은 제3장까지의 내용을 참조할것). 이렇게 하면 Java로 관리하는 이름공간이 같아지므로 패키지유효범위의 메쏘드도 시험할수 있게 된다.

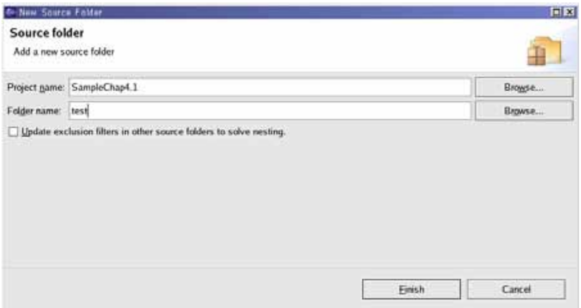


그림 4-5. 새 원천서류철의 생성

4.4.2. 견본프로그람의 설정

그림 4-6과 같이 SampleChap4.2프로젝트를 작성하려고 한다.

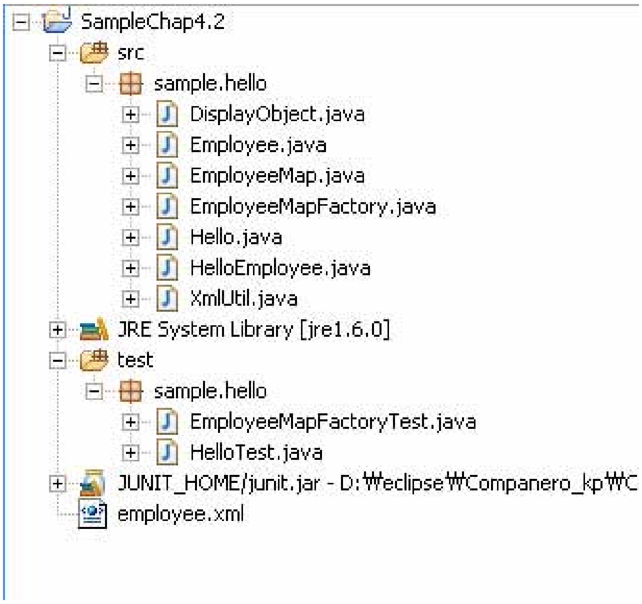


그림 4-6. 견본프로그람의 등록부구성

몇가지 XML용어

앞으로 XML용어를 써서 설명을 하여야 하기때문에 가장 기초적인 XML용어에 대하여 해설한다. 아래에서는 그림자부분이 용어에 대응하고있다.

- 요소(element)

<name> 김강철 </name>

- 요소이름(element name)

<name> 김강철 </name>

- 태그(tag)

<name> 김강철 </name>

- 요소내용(element content)

<name> 김강철 </name>

- 속성(attribute)

<Employee id = “01” >

- 속성이름(attribute name)

<Employee id = “01” >

- 속성값(attribute balue)

<Employee id = “01” >

4.4.3. 견본프로그람을 리용한 시험경우의 작성과 메쏘드의 삽입

여기서 시험경우의 작성레로 리용하려는것은 EmployeeMapFactory클라스의 public EmployeeMap createEmployeeMap()메쏘드이다. 목록 4-9를 보면 이 메쏘드는 아직 삽입되여있지 않다. 이 절에서는 이 메쏘드를 시험구동방식으로 삽입한다.

먼저 메쏘드의 명세서를 확인한다. 명세서는 아래와 같다.

> 이 메쏘드는 XML문서를 읽어들이고 Employee요소가 가지는 속성이름, ID의 속성값과 Employee클라스의 구체레를 한조로 하는 기억기구조를 생성한다.

이 메쏘드는 점차적으로 삽입해가기로 한다.

목록 4-9. createEmployeeMap메쏘드가 삽입전의 EmployeeMapFactory클라스

```
package sample.hello;

public class EmployeeMapFactory {

  private String directoryPath;

  /* 기정구성자는 다른 클라스로부터 보이지 않게 한다 */
  private EmployeeMapFactory() {
  }

  public EmployeeMapFactory(String directoryPath) {
    this.directoryPath = directoryPath;
  }

  public EmployeeMap createEmployeeMap() {
    // 이 메쏘드를 삽입해 나간다
    return null;
  }
}
```

이 createEmployeeMap메쏘드를 삽입하기 전에 시험경우의 메쏘드를 정리해놓는다.

4.4.4. 시험용메쏘드의 개괄

TestCase클라스에는 표 4-1에 제시한 메쏘드가 있다. 여기서 가장 자주 리용되는 메쏘드는

- assertEquals(String str1, String str2)
- fail()

인 2개의 메쏘드이다. assertTrue, assertFalse메쏘드도 있지만 기대값과 메쏘드실행결과값의 차이에 대하여 표시해주지 않는다. 그러므로 기대값과 메쏘드의 실행결과의 차이를 알려주는 assertEquals메쏘드를 사용하는 경우가 많다.

문자렬을 되돌이값으로 돌려주는 메쏘드에 대해서는 JUnit가 많은 비중을 차지한다. 그것은 assertEquals메쏘드가 거의 문자렬을 취급하는 메쏘드이기때문이다. 반대로 문자렬을 되돌이값으로 하지 않는 메쏘드(례하면 GUI나 웨브관련의 메쏘드 등)는 JUnit에서의 시험이 곤난한것으로 된다. 한편 웨브환경으서의 시험을 가능하게 하는 HHPUnit, 자료기지환경에서의 시험을 간편하게 하는 DBUnit 등 JUnit를 확장한 시험기들의 보급도 진척되고 있다.

표 4-1. TestCase메쏘드

메쏘드이름	설 명
assertEquals([인수1, 인수2] 혹은 [인수1, 인수2, 인수3])	기대되는 값인 [인수1]이 시험하는 메쏘드의 결과값들인 [인수2], [인수3]과 같은가를 판정한다. 같을 때에는 시험은 성공한것이다. 또한 기대값과 결과값이 같지 않을 때는 차이점을 지적한다.
assertTrue([인수1] 혹은 [인수1, 인수2])	기대되는 값 [인수1]이 시험하는 메쏘드의 결과값인 [인수2]와 같은가를 판정한다. 같을 때는 시험은 성공한것이다. 그러나 기대값과 결과값이 같지 않을 때 차이점을 지적하지 않는다.
assertFalse([인수1] 혹은 [인수1, 인수2])	기대되는 값인 [인수1]이 시험하는 메쏘드의 결과값인[인수2]와 같은가를 판정한다. 같지 않을 때는 시험은 성공한것이다. 그러나 기대값과 결과값이 같지 않을 때 차이점을 지적하지 않는다.
fail([인수1])	시험을 실패시킨다.
assertNull([인수1] 혹은 [인수1, 인수2])	인수로 지적된 결과값이 null인가를 판정한다. null일 때 시험은 성공한것이다.
assertNotNull([인수1] 혹은 [인수1,인수2])	인수로 지적된 결과값이 null인가를 판정한다. null이 아닐 때 시험은 성공한것이다.
assertSame([인수1] 혹은 [인수1, 인수2])	인수로 지적된 객체가 같은 객체인가를 판정한다. 같을 때 시험은 성공한것이다.
assertNotSame([인수1 혹은 [인수1, 인수2])	인수로 지적된 객체가 같은 객체인가를 판정한다. 같지 않을 때 시험은 성공한것이다.

4.4.5. 시험의 준비와 후처리(setUp, tearDown)

TestCase클라스에는

```
public void setUp()
public void tearDown()
```

이라는 2개의 메쏘드가 준비되여있다. 이 메쏘드는 시험경우클라스안의 각 시험메쏘드의 개시와 완료때마다 호출된다.

례하면 어떤 시험경우클라스의 시험메쏘드

```
testMethod1()
testMethod2()
```

가 있다고 하자. 이 시험경우클라스안에서 setUp메쏘드와 testDown메쏘드를 삽입한 경우

```
setUp()
testMethod1()
tearDown()
setUp()
testMethod2()
tearDown()
```

의 순서로 실행된다. 시험메쏘드의 개시와 완료에서 실행해야 할 어떤 처리가 있을 때에는 처리내용을 setUp메쏘드와 tearDown메쏘드에 삽입하여 리용한다. 례하면 자료기지와의 련결을 setUp메쏘드로 받아놓고 tearDown메쏘드로 반드시 련결을 닫는것과 같은 사용실례를 들수 있다. 이 메쏘드들을 현재 실례에서는 사용하지 않는다.

4.4.6. 편리한 문자렬화(toString메쏘드를 삽입)

준비가 된 조건에서 EmployeeMapFactory클라스의 createEmployeeMap메쏘드의 삽입과 그 시험경우 (EmployeeMapFactoryTest클라스의 testCreateEployeeMap메쏘드)를 서술하기로 한다.

시험대상으로 되는 createEmployeeMap메쏘드의 되돌이값의 형이 EmployeeMap로 되여있다. 되돌이값인 EmployeeMap클라스의 구체레로부터 종업원의 ID값을 열쇠로 하여 종업원정보를 표시하는 Employee클라스의 구체레를 얻을수 있다. 그러므로 시험방법은 종업원의 ID값을 열쇠로 하여 Employee클라스의 구체레를 얻은 다음 기대한 종업원정보인가를 확인하는 방법이 좋다. Employee클라스의 구체레를 얻은 다음 기대한 결과와 일치하는가를 평가하자면 Employee클라스의 Getter메쏘드를 써서 매번 assertEquals메쏘드로 기대값과 비교하면 된다.

목록 4-10. 도중에서 실패하는 경우

```
package sample.hello;
import junit.framework.TestCase;
public class EmployeeMapFactoryTest extends TestCase {
    /**
    * Constructor for EmployeeMapFactoryTest.
    * @param arg0
    */
    public EmployeeMapFactoryTest(String arg0) {
        Super(arg0);
    }
    public void testCreateEmployeeMap() throws Exception {
        // 시험경우메쏘드를 서술한다.
        EmployeeMapFactory eFactory = new EmployeeMapFact
        ory( "employee.xml" );
        employeeMap eMap = eFactory.createEmployeeMap();
        Employee e01 = eMap.getEmployee( "01" );
        // .....
    }
}
```

여기서 매번 Getter메쏘드를 리용하여 비교하는 방법도 좋지만 Getter메쏘드의 수가 많으면 코드가 복잡해진다. JUnit에서 어떻게 assertEquals메쏘드를 리용하는가가 매우 중요하다는것은 이미 강조하였다. 이 실례에서처럼 마당의 수가 많고 그에 비례하여 Getter메쏘드가 많아지는 클라스의 구체례를 되돌이값으로 하는 메쏘드를 시험하는 경우 다음과 같은 수법이 자주 사용된다.

> 객체의 문자렬화형식을 써서 결과를 평가한다. 즉 toString메쏘드를 써서 결과를 평가한다.

모든 클라스는 직접적으로나 간접적으로나 java.lang.Object클라스(아래에서는 Object클라스로 략칭한다.)를 계승하고있다. 그리고 Object클라스는 toString메쏘드를 삽입하고있으므로 필연적으로 모든 클라스는 toString메쏘드를 삽입하고있는것으로 된다. 그러나 Object클라스에서 정의하고있는 toString메쏘드는 Java가상기계가 할당한 구체례의 ID값을 표시하기만 한다. ID값을 표시하기만 해서는 시험경우에서의 평가에는 쓸수 없다.

그래서 Employee클라스에서 toString메쏘드를 재정의(override)하여 사용하기로 한다. toString메쏘드의 삽입례를 목록 4-11에서 보여주었다.

목록 4-11. Employee클라스에 추가한 toString메소드(Employee.java)

```
/**
 * 이 객체를 문자렬화한다.
 */
public String toString() {
return "["
    + id + ","
    + name + ","
    + role + ","
    + telNumber + ","
    + boss
    + "]";
}
```

이 toString메쏘드를 사용하면 객체는 [김영철 , 실장, 999888777, null]와 같은 결과를 되돌린다. 여기서 실례로 든것은 Employee클라스가 보존하는 마당값을 반점으로 구분하여 라렬한것이지만 XML형식으로 결과를 되돌리는것도 좋은 방법이다. 목록 4-12에 객체의 문자렬화형식을 리용하여 작성한 시험경우를 보여주었다.

목록 4-12. 객체의 문자렬화를 리용한 시험경우(EmployeeMapFactoryTest.java)

```
package sample.hello;
import junit.framework.TestCase;
public class EmployeeMapFactoryTest extends TestCase {
    /**
    * Constructor for EmployeeMapFactoryTest
    * @param arg0
    */
    public EmployeeMapFactoryTest(String arg0) {
        super(arg0);
    }

    // Employee클라스의 toString메쏘드를 리용한 시험경우의 삽입
    public void testCreateEmployeeMap() throws Exception {
        EmployeeMapFactory eFactory = new EmployeeMapFactory
     ("employee.xml");
        EmployeeMap eMap = eFactory.createEmployeeMap();
        Employee e01 = eMap.getEmployee("01");

        // 문자렬화하고 검사
        assertEquals("[01, 김영철, 실장, 999888777, null]",
     e01.toSting());
    }
}
```

4.4.7. 삽입의 완성

시험경우 메쏘드를 삽입했으므로 고의적으로 성공시키는 경우에 대하여 보기로 한다.

목록 4-13. 고의적으로 성공시키는 메쏘드의 삽입(EmployeeMapFactory.java)

```
package sample.hello;
import java.io.File;
import java.util.HashMap;
public class EmployeeMapFactory {
    /* 기정구성자는 다른 클라스로에서 보이지 않게 한다. */
    private EmployeeMapFactory() {
    }
    private String directoryPath;
    public EmployeeMapFactory(String directoryPath) {
        this.directoryPath = directoryPath;
    }

    // 고의적으로 시험을 성공시키는 메쏘드의 삽입
    public EmployeeMap createEmployeeMap() {
        java.util.Map employeeMap = new HashMap();
        File xmlFile = new File(directoryPath);
        Employee emp
        = new Employee( "01" , "김영철" , "실장" , 999888777);
        employeeMap.put( "01" ,emp);
        return new EmployeeMap(employeeMap);
    }
}
```

시험의 시작단계에서는 이처럼 메쏘드를 간단히 삽입함으로써 이후의 본격적인 코드작성에 쓸만한 메쏘드의 삽입방향을 가늠할수 있다. 즉

XML문서로 서술되여있는 Employee요소의 수만큼 Employee클라스의 구체레를 생성한다. 생성한 구체레로부터 종업원ID(employeeID)를 하쉬열쇠로 하여 하쉬값령역(hash map)에 넣는다. 마지막으로 그 하쉬값령역을 EmployeeMap클라스의 구성자인수로 하여 객체를 생성한다.

DOM나 SAX와 같은 XML의 API에 익숙되여있지 않은 경우 코드작성이 힘들것이다. 여기서는 XML문서로부터 요소내용이나 속성값을 꺼내고 Employee클라스의 구체레를 생성하는 처리가 어떤 틀거리를 가지는가를 리해하면 된다. 한번도 써본적이 없는 API를 쓸 때는 이처럼 시험을 진행하면서 삽입하면 속도는 떨어지지만 정확히 리용할수 있다.

4.4.8. 여러단계시험

마지막으로 여러단계시험을 진행하여 실지로 createEmployeeMap메쏘드가 삽입되여있는가를 확인한다. 삽입례는 목록 4-14와 같다.

목록 4-14. 여러단계시험을 통한 시험경우의 실현

```
public void testCreateEmployeeMap() throws Exception {
    EmployeeMapFactory eFactory =
              New EmployeeMapFactory( “employee.xml” );
    EmployeeMap eMap = eFactory.createEmployeeMap();
    Employee e01 = eMap.getEmployee( “01” );
    // 문자렬화하고 검사
    assertEquals( “[01, 김영철, 실장, 999888777, null]” , e01.toString());
    // 여러단계시험1
    Employee e02 = eMap.getEmployee( “02” );
    assertEquals(  “[02, 백남진, 설계원, 111122223,”
        + “[01, 김영철, 실장, 999888777, null]” , e02.toString());
    // 여러단계시험2
    Employee e03 = eMap.getEmployee( “03” );
    assertEquals(  “[03, 김철혁, 기사, 666,”
        + “[01, 김영철, 실장, 999888777, null]” , e03.toString());
}
```

이상으로 시험경우가 완성되였다. 시험경우를 실행하여 록색띠가 나타나는가를 확인해본다.

4.4.9. 시험을 묶어서 실행

지금까지 작성한 시험경우를 묶어서 실행하기 위한 방법에 대하여 소개한다.

[Package Explorer]안의 [SampleChap4.3]프로젝트아래의 [test]등록부에 있는 [sample.hello]패키지를 선택하고 오른쪽찰칵한다. [New]→ [Other]를 선택하여 현시되는 대화칸의 [Java]→[JUnit]에서 [JUnit Test Suite]를 선택한다. [Next]단추를 찰칵하고 시험묶음의 이름을 《 AllTests 》라고 입력한다. 마지막으로 EmployeeMapFactoryTest와 HelloTest를 선택하고 [Finish]단추를 찰칵한다(그림 4-7). 이상의 절차에 의하여 시험경우를 묶은 AllTests라는 이름의 시험묶음클라스를 Eclipse가 자동생성하여 준다.

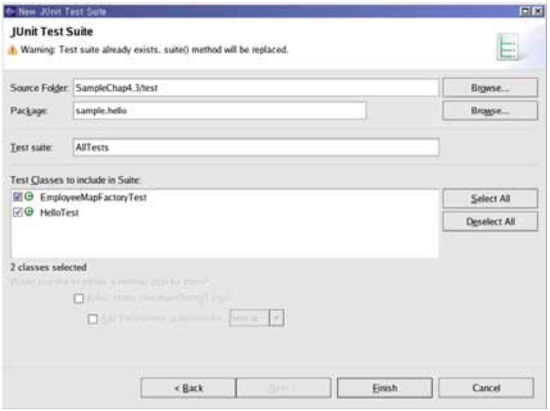


그림 4-7. 시험묶음작성

이 시험묶음(AllTests)을 실행하는 단계는 아래와 같다.

[Package Explorer]안의 AllTests.java를 오른쪽찰칵→[Run]→ [JUnit Test]을 선택하여 실행한다. 실행하면 프로젝트안의 모든 시험경우가 자동실행된다.

Eclipse3.0에서 시험경우의 처리

Eclipse2.0까지의 판본에서는 시험경우들을 묶어서 실행하기 위한 시험묶음을 작성할 필요가 있었다. 그러나 Eclipse2.1에서부터는 자동적으로 프로젝트안에 존재하는 시험경우에 대하여 모두 실행하게 되였다.

Eclipse3.0에서도 마찬가지이다. [Package Explorer]안의 시험을 진행하려는 프로젝트이름에 유표를 맞춘다. 다음으로 유표를 맞춘 프로젝트이름에서 오른쪽찰칵→[Run]→[JUnit Test]을 선택하면 프로젝트안의 모든 시험경우를 실행한다.

제5절. Eclipse를 사용한 시험작성의 요점

JUnit를 사용하면 여러가지 시험경우에 대하여 작성할수 있다. 례하면 JDBC의 대용체(stub)를 써서 실지로 자료기지와 접속하지 않아도 시험을 진행할수 있는 환경을 만들어놓고 JUnit를 쓸수도 있다. 이러한 JDBC와 같은 대면부의 간단한 대용체를 일반적으로 **위조객체**(mock)라고 부른다. 시험경우를 작성함에 있어서 위조객체를 활용하는것이 매우 중요하다.

또한 JUnit를 잘 리용하려면 경험과 부단한 학습이 동반되여야 한다. 시험의 문제포괄정도와 견고성을 고려하려면 쏘프트웨어공학에서의 시험기법에 익숙해야 한다. JUnit의 우점은 초기의 학습에서 시험에 필요하다고 인정되는 기술중에서 60%정도 소유할수 있는것이다. 또한 JUnit를 리용하면 목표를 작게 분할하고 JUnit를 적용할수 있는것부터 시작할수 있다.

JUnit의 효능

JUnit가 편리한것은 사실이고 품질도 1.25배~1.6배정도로 높아지지만 JUnit를 리용하지 않고 그냥 코드를 작성할 때에 비하여 공수는 2.5배정도로 늘어난다. 개편할 때 매우 편리하고 Junit를 사용한 회귀시험에 의하여 오유를 막을수 있다는 우점과 함께 공수가 많이 든다는 결함도 있다는것을 알아야 한다.

일반적으로 틀거리개발과 같은 개발인원수가 작고 자주 변경할 필요가 있는 프로젝트에서는 JUnit의 효과가 크며 업무체계와 같은 개발인원을 많이 요구하는 프로젝트에서는 공수문제를 무시할수 없기때문에 일부 중요 부문에만 적용하는것이 좋다.

이것은 소규모고품질체계를 개발할 때는 일정한 능력이상의 많지 않는 인원이 충분한 정보교환을 보장하는것이 가장 효과적이라고 주장하는 XP수법의 요구와 일치한다. 한편 대규모업무프로젝트에서는 우선 많은 인원을 확보해야 하기때문에 자연히 개발성원들의 기능수준이 서로 같지 않으며 따라서 의사소통을 충분히 보장하기가 힘들어진다. 그리하여 이와 같은 대규모프로젝트에서 XP수법을 적용하기가 곤난하다.

제5장. 오유수정

4장에서 설명한바와 같이 JUnit를 사용하여 시험구동개발을 진행하는것으로 오유가 적은 효률높은 프로그람을 작성할수 있다. 그렇지만 처음부터 오유가 전혀 없는 프로그람을 작성할수는 없다. 특히 개발과정에서는 부주의나 실수로 인하여 오유가 나타나게 되며 다음과 같은 문제들이 발생할수 있다.

- 시험이 실패하였으나 Junit의 통보문만으로는 원인을 잘 알수 없다.
- 시험에서 빠진 검사항목이 있어 프로그람이 예상외의 동작을 한다.
- 실행할 때 례외가 발생하여 프로그람이 갑자기 정지해버린다.

이러한 때에는 오유수정프로그람을 사용한다. Eclipse에는 오유수정을 지원하는 여러가지 기능이 갖추어져있다.

제1절. 오유수정프로그람

5.1.1. 오유수정프로그람의 기동

오유수정프로그람을 기동하는 순서는 프로그람의 실행순서와 거의 같다.

다음과 같은 3가지 방법이 있다.

1. 차림표띠에서 [Run]→[Debug History]을 선택하여 현시되는 대화칸에서 기정설정을 그대로 선택한다.
2. [Package Explorer]보기에서 오유수정하려는 클라스나 서류철을 선택하고 차림표띠에서 [Run]→[Debug As]를 선택한다. 여기서 Java Application, Java Applet, JUnit Test, JUnit Plug-in test, Runtime Work Bench중에서 필요한것을 선택한다.
3. 차림표띠에서 [Run]→[Debug]를 선택하면 현시되는 Debug대화칸(그림 5-1)에서《configurations:》목록의 필요한 항목을 선택한 다음 [Debug]를 찰칵한다.

그림 5-1의 대화칸에서 기동구성의 종류별로 투시와의 련관을 맺어줄수 있다. 기정으로는 어떤 기동구성을 설정해도 오유수정프로그람을 기동하여 멈춤점(break point)에서 실행을 중단하면 [Debug]투시로 설정이 바뀌여진다.

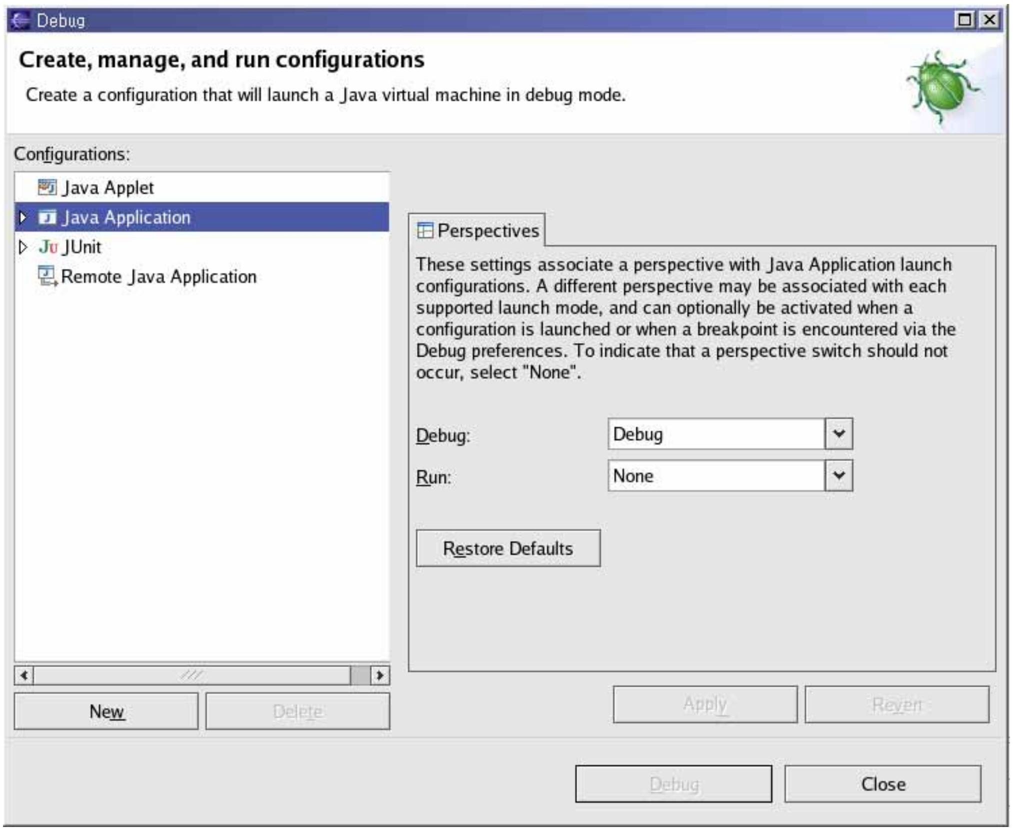


그림 5-1. 기동구성과 투시와의 관련설정

차림표띠외에 도구띠의 [Debug]단추(벌레모양의 아이콘)나 [Package Explorer]보기, 또는 [Outline]보기를 오른쪽찰칵하여 현시되는 지름차림표로부터도 오유수정프로그람을 기동할수 있다. 또한 오유수정프로그람의 기동과 프로그람의 실행에서 같은 기동구성을 리용할수도 있다. 기동구성의 자세한 내용에 대하여서는 3장의 제3절을 참조하기로 한다.

실지 오유수정프로그람을 기동하여 보기로 한다. Eclipse를 기동하고 SampleChap5프로젝트를 연다. 여기서는 오유수정용으로 새롭게 기동구성을 작성해본다.

[Java]투시로 절환하여 차림표띠에서 [Run]→[Debug]을 선택하여 Debug대화칸을 연다. 대화칸의 왼쪽에서 [Java Application]을 선택하여 [Perspectives]태브의 [Debug]의 내리펼침목록에서 [Debug]가 선택되였는가를 확인한 후 [New]단추를 찰칵한다. 다음 대화칸의 오른쪽에 있는 [Name]에 《Hello Employee Debug》, [Project]에 《SampleChap5》, [Main Class]에 《sample.hello.HelloEmployee》를 지정한다(그림 5-2). [Stop in main]검사칸을 선택하여 놓으면 main메쏘드가 호출

되는 곳에서 스레드가 중단된다. 이 검사칸을 선택하여 [Debug]단추를 찰칵한다. 투시를 절환하겠는가 하는 확인을 요구하는 대화칸(그림 5-3)이 현시되는데 다음부터는 이 확인통보문이 필요없으면 대화칸에서 [Remember my decision]검사칸을 선택한다. 대화칸의 [Yes]단추를 찰칵하여 오유수정프로그람을 기동한다.

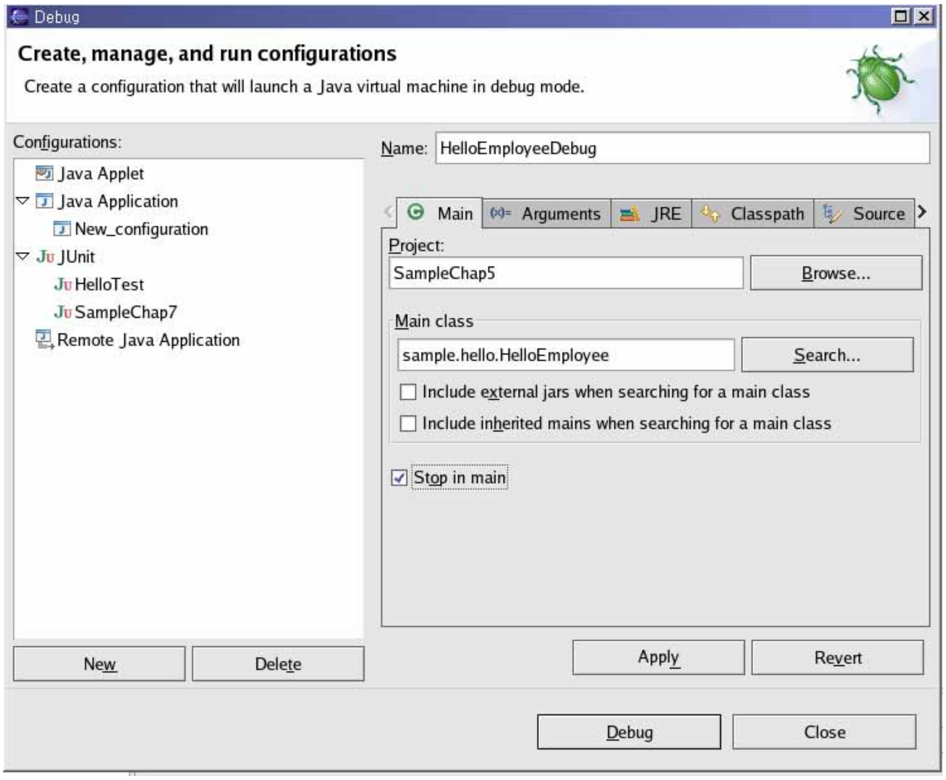


그림 5-2. 오유수정프로그람의 기동구성작성

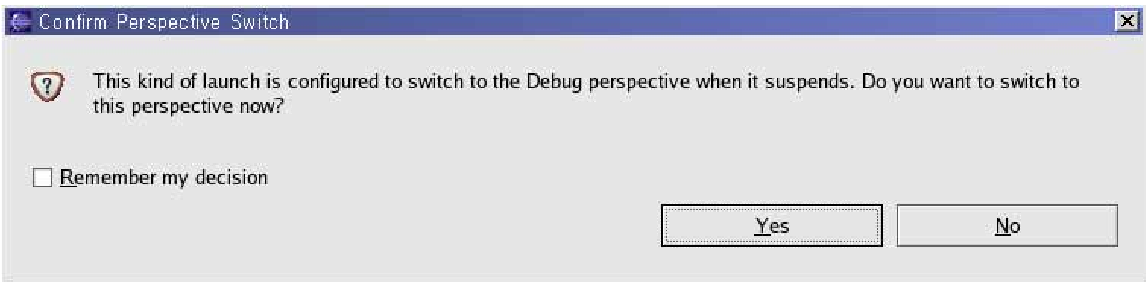


그림 5-3. 투시절환의 확인대화칸

5.1.2. [Debug]투시

오유수정프로그람이 기동하면 그림 5-4와 같은 [Debug(오유수정)]투시가 열린다. [Debug]투시에는 오유수정에 필요한 조작의 실행이나 정보의 참조에 해당한 보기들이 있다.

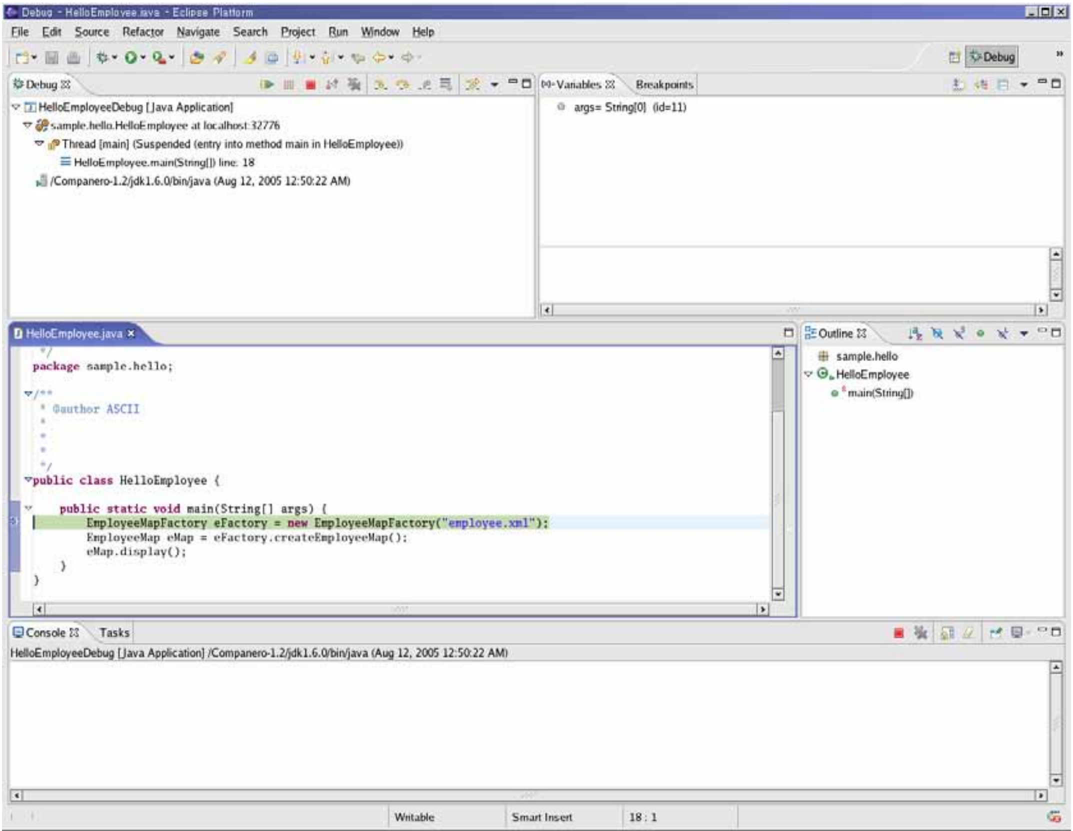


그림 5-4. [Debug]투시

웃부분의 좌측에 있는것이 [Debug]보기이다. [Debug]보기에는 현재의 스레드 및 탄창프레임(stack frame)이 현시된다. 기동구성에서 main메쏘드에서 중단하도록 설정한 결과 스레드가 중단된것을 알수 있다. [Debug]보기의 웃부분에는 단추가 있고 멈춤점에서 중단되는 스레드를 다시 열거나 실행중의 스레드를 일시 중단시킬수도 있고 완전히 정지시키는 조작이 가능하다. 오유수정에서 중요한 걸음(Step)실행도 [Debug]보기의 단추를 사용하여 진행된다. 오른쪽우에 [Variables]보기, [BreakPoints(멈춤점)]보기가 있다. 이런 보기들에 대한 구체적인 내용은 후에 서술하겠지만 여기서는 변수값의 확인이나 변경, 멈춤점의 유효화, 무효화 등이 진행된다.

[Debug]투시의 가운데부분에는 Java편집기와 [outline]보기가 있다. 멈춤점에서

중단하면 Java편집기에 현재 실행중의 원천코드가 표시되고 중단되여있는 멈춤점의 위치가 화살표로 표시된다. 또한 멈춤점을 설정하지 않았으나 기동구성의 설정에서 [main에서 정지]를 선택하였으므로 주메쏘드(main method)의 제일 첫 행에서 중단한다.

아래부분에는 [console]보기와 [task]보기가 있다. 오유수정프로그람이 기동할 때에 기동구성에서 JUnit를 선택한 경우는 [JUnit]보기가 추가된다.

이와 같이 [Debug]투시의 기본적인 보기에 대하여 소개하였다. 실지 오유수정에서는 이런 보기들을 사용하여 여러가지 조작을 진행한다. 지금은 프로그람의 실행이 중단된 상태이지만 여기서 [Debug]보기의 [Resume]단추(그림 5-5)를 찰칵해보면 중단되였던 스레드가 재시동되며 나머지 처리가 실행되게 된다. 결과 프로그람이 정상적으로 완료된다.

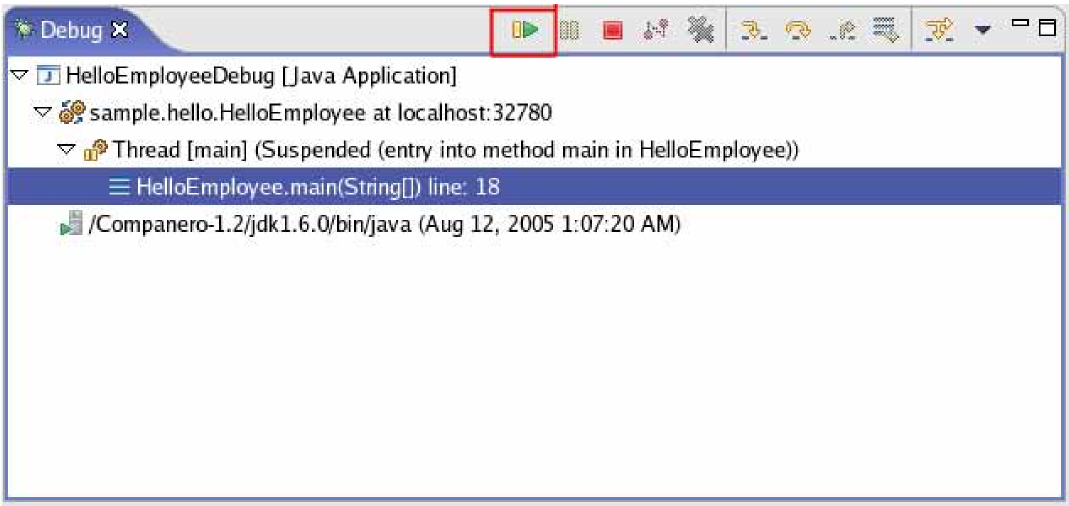


그림 5-5. [Debug]보기

제2절. 멈춤점

5.2.1. 멈춤점의 개요

멈춤점을 설정하여 오유수정프로그람을 기동하면 멈춤점이 있는 위치에서 프로그람 실행이 중단된다. 중단하고있는 동안은 그 시점에서의 변수값을 조사하고 식의 값을 평가하는 등 오유수정에 필요한 정보를 얻을수 있다.

Eclipse에서 설정할수 있는 멈춤점을 표 5-1에 주었다.

표 5-1. 멈춤점의 종류

종 류	설 명
Line Breakpoint	메쏘드내의 원천코드행에 대해 설정하는 멈춤점. 그 행의 코드를 실행하기직전에 실행을 중단시킬수 있다.
Method Breakpoint	메쏘드에 대하여 설정하는 멈춤점. 메쏘드의 불러내기(입구점)나 메쏘드의 되돌림(완료)에서 실행을 중단시킬수 있다.
Watchpoint	마당에 대하여 설정하는 멈춤점. 마당의 호출이나 마당에로의 대입(변경)에서 실행을 중단시킬수 있다.
Java Exception Breakpoint	례외의 형을 지정하여 설정하는 멈춤점. 포착되는 례외나 포착되지 않는 례외를 버릴 때 실행을 중단시킬수 있다.
Class Breakpoint	클라스를 지정하여 설정하는 멈춤점. 해당한 클라스가 제일 처음으로 적재될 때 실행을 중단시킬수 있다.

5.2.2. 행멈춤점의 설정

[Package Explorer]보기에서 《SampleChap5》 프로젝트 →[src]서류철→[sample.hello]패키지를 전개하여 EmployeeMapFactory.java를 두번 찰칵하면 원천코드를 편집기에 현시한다. LoadEmployee메쏘드의 가운데부분에 있는 for문의 다음행에 《Node childNode=nodeListitem(i);》라는 행이 있는데 여기에 행멈춤점을 설정한다. 이행의 좌측에서 표식띠(marker bar)를 오른쪽찰칵하여 [Toggle Line Breakpoint]을 선택하면 그림 5-6과 같이 표식띠에 청색의 동그란표식이 붙여진다. 이 표식은 행멈춤점이 유효하다는것을 나타낸다.

설정한 행멈춤점을 제거하려면 행멈춤점의 표식을 오른쪽찰칵하여 [Toggle Line Breakpoint]을 선택한다. [Toggle Line Breakpoint]이 아니라 [Disable Breakpoint] 를 선택하면 청색표식이 흰색표식으로 변경되여 일시적으로 행멈춤점이 무효로 된다. 사용할수 없게 된 행멈춤점도 [Debug]투시의 [BreakPoints]보기에서는 목록으로 현시되기때문에

멈춤점의 유효/무효를 자주 절환하려는 경우에는 없애버리지 않고 사용불가능으로 해놓으면 편리하다.

사용불가능한 행멈춤점을 사용가능하게 하려면 행멈춤점표식을 오른쪽찰칵하여 [Enable Breakpoint]를 선택한다. 여기서 설명한 멈춤점의 제거 및 사용가능/사용불가능의 절환조작은 설정시의 메쏘드멈춤점이나 감시점에 대하여도 똑같이 진행된다.

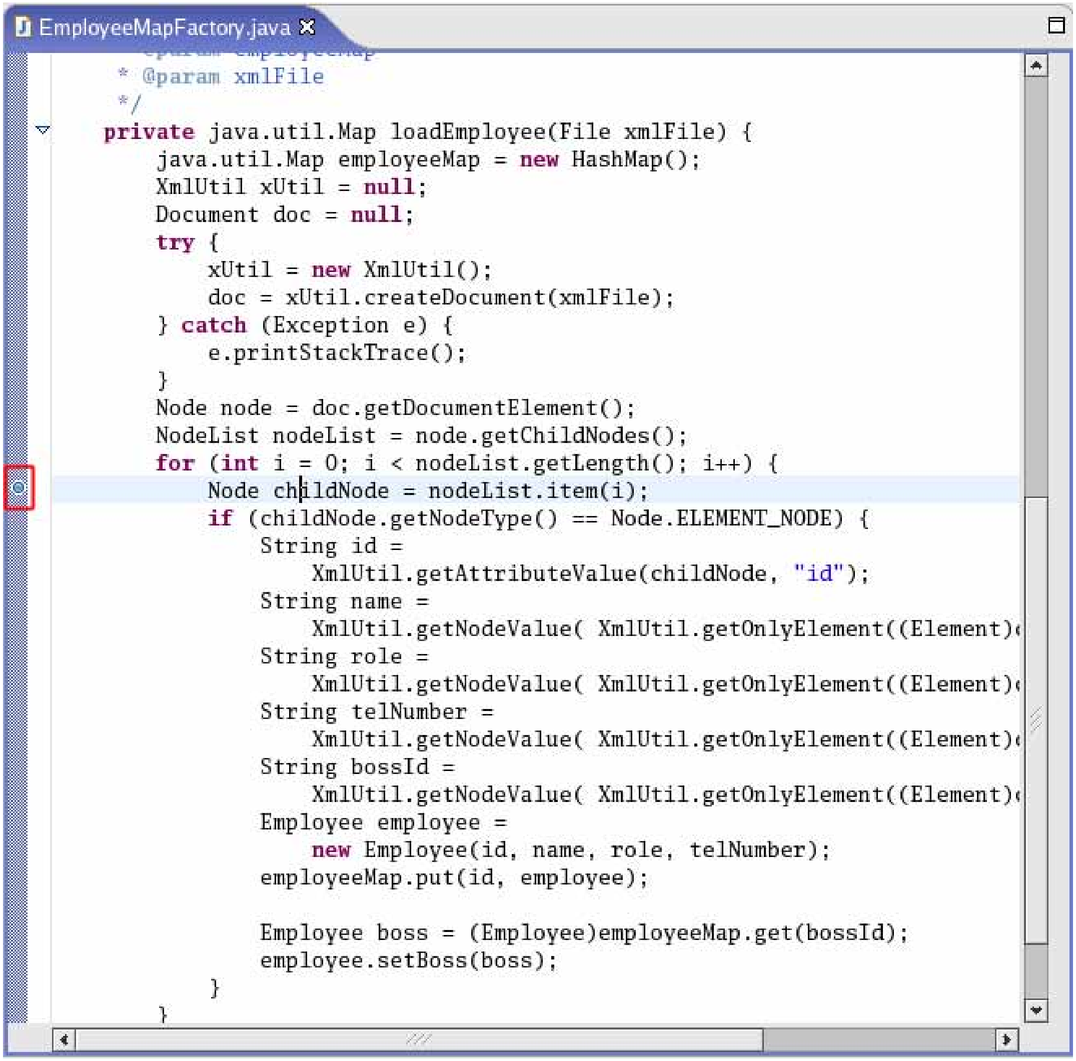


그림 5-6. 행멈춤점

행멈춤점의 표식을 오른쪽찰칵하여 표시되는 지름차림표로부터 [break point property]을 선택하면 그림 5-7과 같은 대화칸이 현시되고 행멈춤점의 자세한 속성을

설정할수가 있다. 이 대화칸에서도 행멈춤점의 사용가능/사용불가능을 절환할수 있다. 그러면 앞에서 설정된 행멈춤점에 명중계수기(hit counter)를 설정해본다. 그림 5-7의 대화칸에서 [common]을 선택하고 [hit Count:]검사칸을 선택한 다음 본문칸에 《2》를 입력한다. 그러면 두번째로 그 행이 실행되였을 때만 멈춤점에서 스레드가 중단되게 된다. 이런 행멈춤점은 for순환의 내부에 있기때문에 순환의 두번째 실행시에 중단되게 된다.([5.3.1. 멈춤점에서의 중단과 재개]에서 확인한다.) [OK]단추를 찰칵하여 대화칸을 닫는다.

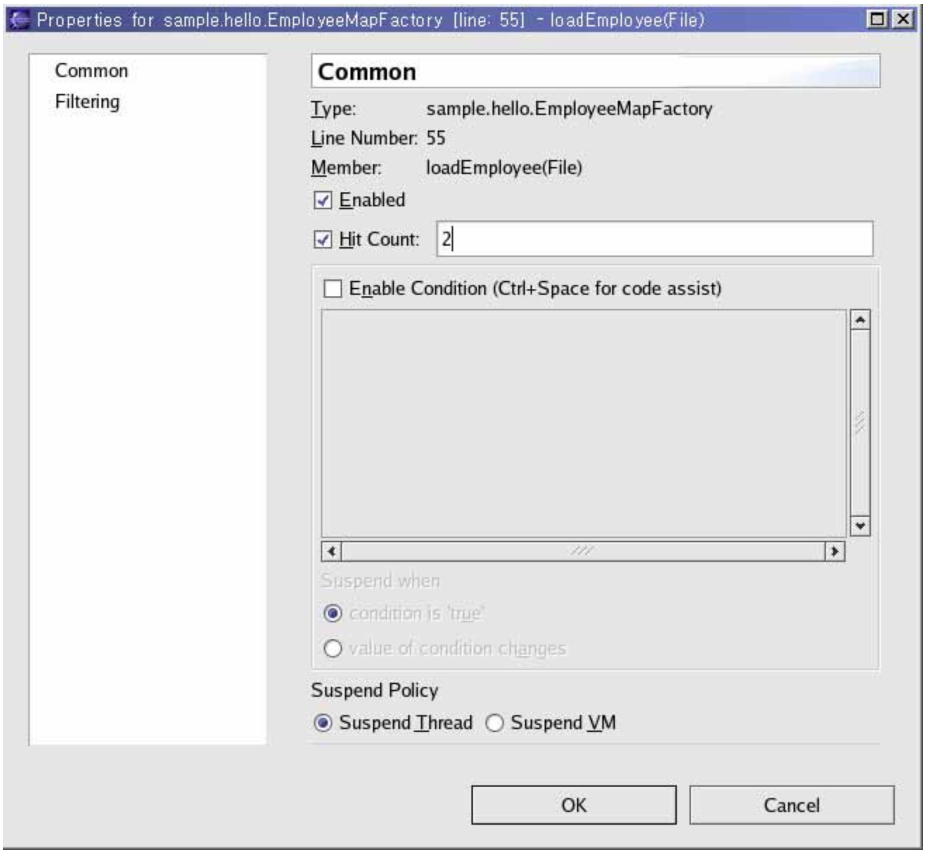


그림 5-7. 행멈춤점의 속성설정

그리고 행멈춤점을 설정 혹은 제거하기 위해서는 편집기의 표식띠를 두번 찰칵한다. 설정과 제거만 있으면 이런 조작들이 간단하지만 사용가능/사용불가능을 절환하려면 앞부분에서 설명한 순서가 필요하다는데 대해 주의를 돌려야 한다.

5.2.3. 메쏘드멈춤점의 설정

메쏘드멈춤점의 설정은 편집기의 표식띠의 지름차림표가 아니라 차림표띠의 [Run] 혹은 [Package Explorer]보기나 [Outline]보기의 지름차림표를 사용한다. 여기서는 차림표띠를 사용하여 메쏘드멈춤점을 설정한다.

1. [Java]투시를 열고 [Package Explorer]보기에서 [SampleChap5]프로젝트→[src]서류철→[sample.Hello]패키지→[XmlUtil.Java]를 두번 찰칵하여 원천코드를 현시한다.
2. [outline]보기에서 createDocument(File)메쏘드를 선택한다.
3. 차림표띠에서 [Run]→[Toggle Method Load Breakpoint]을 선택한다.
4. 원천코드의 메쏘드선언부에 메쏘드멈춤점의 표식이 나타난다.(그림 5-8)

결과 메쏘드멈춤점에 의해 메쏘드를 호출할 때 실행이 중단된다. 메쏘드멈춤점의 표식을 오른쪽찰칵하여 [break point property]을 선택한다. 이때 현시되는 대화칸에서 실행을 중단시킬수 있는 조건을 변경시킬수 있다.

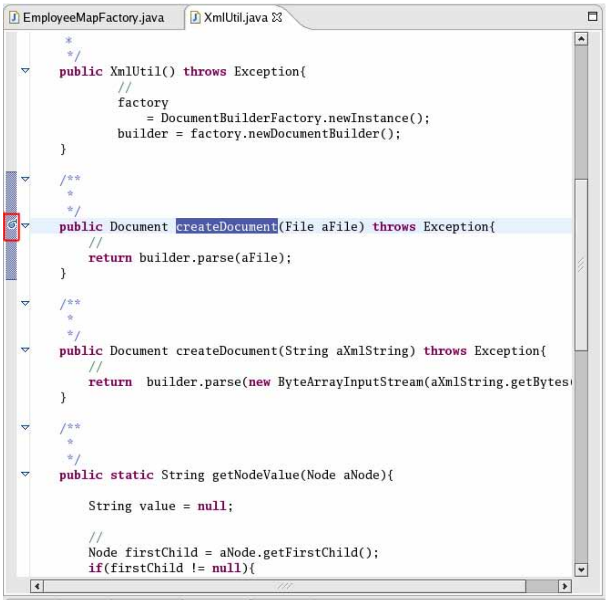


그림 5-8. 메쏘드멈춤점

5.2.4. 감시점의 설정

감시점의 설정도 차림표띠의 [Run] 혹은 [Package Explorer]보기나 [outline]보기의 지름차림표를 사용한다. 여기서는 [outline]보기를 사용하여 설정해본다.

1. [Java]투시를 열고 [Package Explorer]보기에서 [Samplechap5]프로젝트→[src]서류철→[sample.Hello]패키지→Employee.Java]를 두번 찰칵하여 원천

코드를 현시한다.

2. [outline]보기에서 name마당을 오른쪽찰칵하여 현시되는 지름차림표에서 [toggle with Watchpoint]을 선택한다.
3. 원천코드의 마당선언부에 감시점의 표식이 나타난다(그림 5-9).

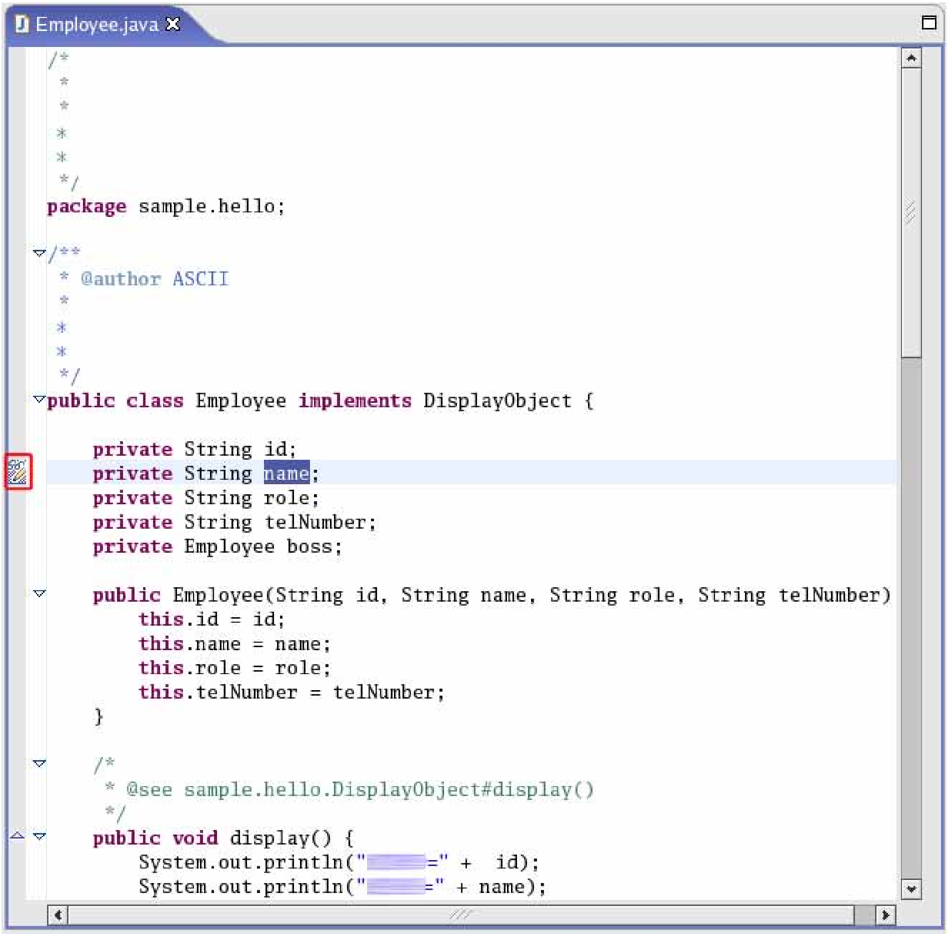


그림 5-9. 감시점

감시점은 기정으로는 마당을 변경하거나 참조할 때 실행을 중단시킨다. 감시점표식을 오른쪽찰칵하여 [break point property]을 선택한다. 이때 현시되는 대화칸에서 실행을 중단시키는 조건을 변경시킬수 있다.

제3절. 오유수정프로그람의 실행

5.3.1. 멈춤점에서의 중단과 재개

그러면 5장의 제2절에서 설정된 멈춤점에서 실행이 중단되는가를 확인해보도록 한다. 차림표띠에서 [Run]→[Debug]를 선택한다. 열려진 대화칸의 왼쪽에 이미 설정한 HelloEmployeeDebug의 기동구성이 있으므로 그것을 선택한다. [5.1.1. 오유수정프로그람의 기동]에서 오유수정프로그람을 기동하였을 때에는 [stop in main]에 선택을 하였으나 이번에는 선택하지 않고 [Debug]단추를 찰칵한다. 제일 처음의 멈춤점에서 실행이 중단되고 [Debug]투시로 절환된다. 결국 이미 설정한 메쏘드멈춤점인 XmlUtil클라스의 createDocument(File)메쏘드의 제일 첫 행에서 중단된다(그림 5-10).

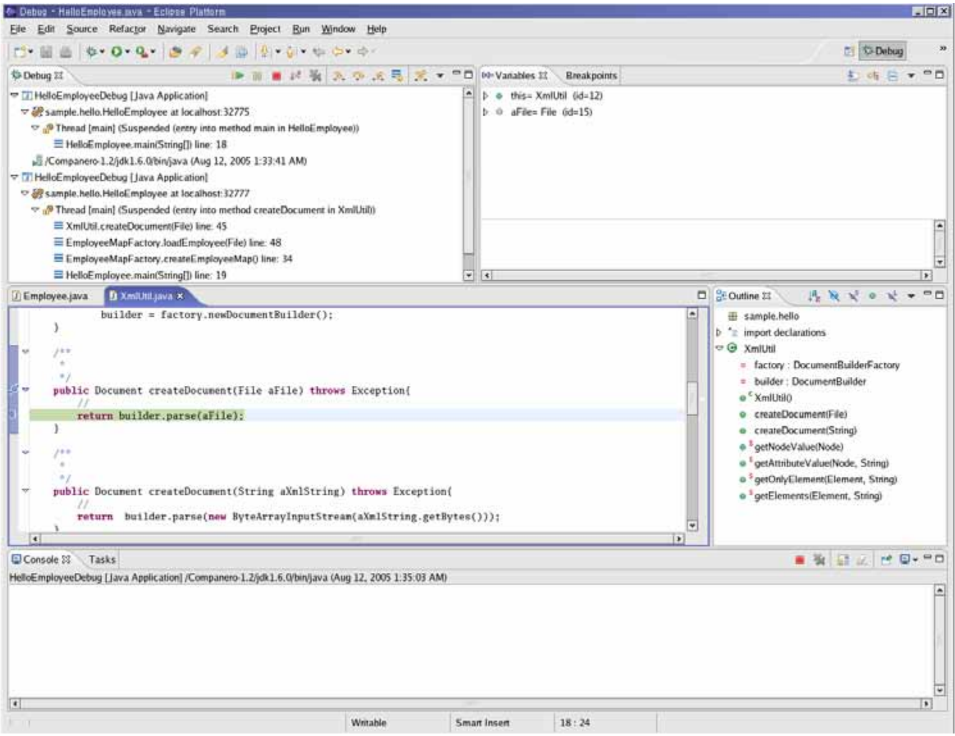


그림 5-10. 메쏘드멈춤점에 의한 중단

중단된 스레드의 재개나 정지 등의 조작은 [Debug]보기의 도구띠단추를 사용한다.(그림 5-11) 단추는 3개의 그룹으로 나뉘여져있는데 왼쪽그룹에 스레드의 [Resume],[Suspend(중단)],[Finish]단추가 있다. 가운데그룹에는 스레드실행을 위

한 3개의 단추가 배치되여있다. [Resume]단추를 사용하면 다음번 멈춤점까지 단번에 실행해버리지만 걸음실행단추를 사용하면 원천코드상에서 한행만 실행하고 다시 실행이 중단된다. 걸음실행에 의하여 실행을 한행씩 따라가면서 변수값의 변화 등을 자세히 추적할수 있다.

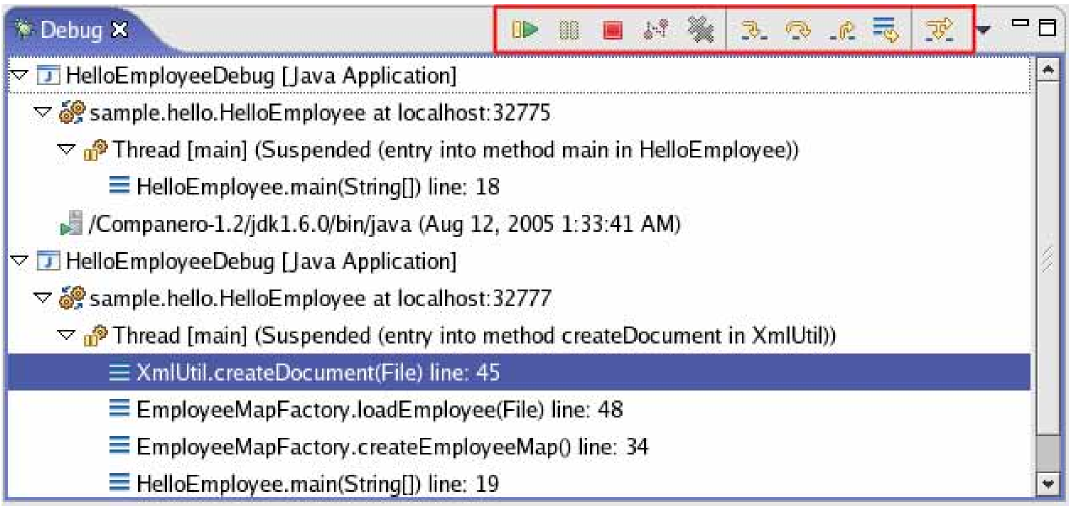


그림 5-11. [Debug]보기의 단추

이제 [Resume]단추를 찰칵하여 다음 멈춤점까지 단번에 실행해본다. 그러면 for순환의 제일 첫 행에 설정한 행멈춤점에서 중단된다(그림 5-12).

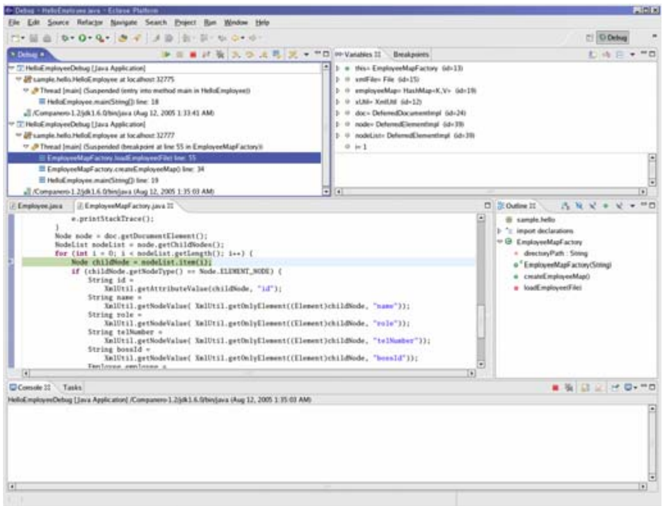


그림 5-12. 행 멈춤점에서의 중단

여기서 순환계수기의 변수 i우에 마우스지적자를 이동시키면 그림 5-13과 같이 변수 i의 값이 풍선도움말로 표시된다. 이렇게 마우스지적자를 변수우에 이동시켜 변수의 내용을 간단히 확인할수 있다. 현재 i의 값이 1로 되여있다. 멈춤점의 속성에서 Hit Count를 2로 설정했기때문에 for순환의 두번째 실행시에 중단되는것을 확인할수 있다.

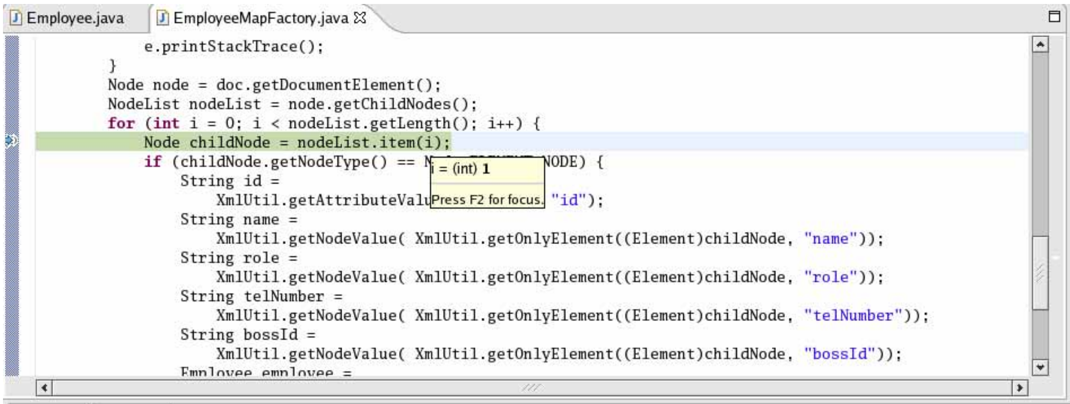


그림 5-13. 변수값의 참조

5.3.2. 걸음바깥(step-over), 걸음안(step-in), 걸음되돌림(step-return)

여기서는 걸음실행을 해본다. 걸음실행에는 3종류의 단추가 있다. 우선 오유수정보기의 [step-over]단추를 찰칵해본다. [step-over]단추를 사용하면 현재 원천코드상에서 한 행만 실행하고 다음 행에서 중단된다. [string role=]행이 반전할 때까지 [step-over]단추를 여러번 찰칵하여 확인하여본다.

다음으로 [step-in]단추를 찰칵한다.
이때 [XmlUtil.GetNodeValue(XmlUtil.getOnlyElemeut((Element)childNode, “telNumber”);)의 행이 반전되는것은 [step-over]단추를 리용하였을 때와 차이가 없다. 또 한번 [step-in]단추를 찰칵하면 이번에는 XmlUtil클라스의 원천코드가 편집기에 현시되며 getOnlyElements메쏘드의 제일 첫 행에서 중단된다(그림 5-14의 2번째 그림).

이와 같이 [step-in]단추를 사용하면 원천코드상에서 다른 메쏘드를 호출한 경우 그 메쏘드의 내부도 걸음실행하는것으로 된다. 다시 한번 [step-in]단추를 찰칵하면 getOnlyElement메쏘드의 내부로 뛰여넘기한다. 다시 [step-in]단추를 찰칵해본다. ParentNode클라스의 getElementsByTagName메쏘드에 뛰여넘기한다. 이 ParentNode클라스는 JRE의 체계서고에 포함되여있는 클라스이다. 이와 같이 [step-in]에 의한 걸음실행은 JRE나 외부jar파일의 원천코드가 첨부되여있으면 그들의 내부까지 걸음실행하여 오유를 수정할수 있다.

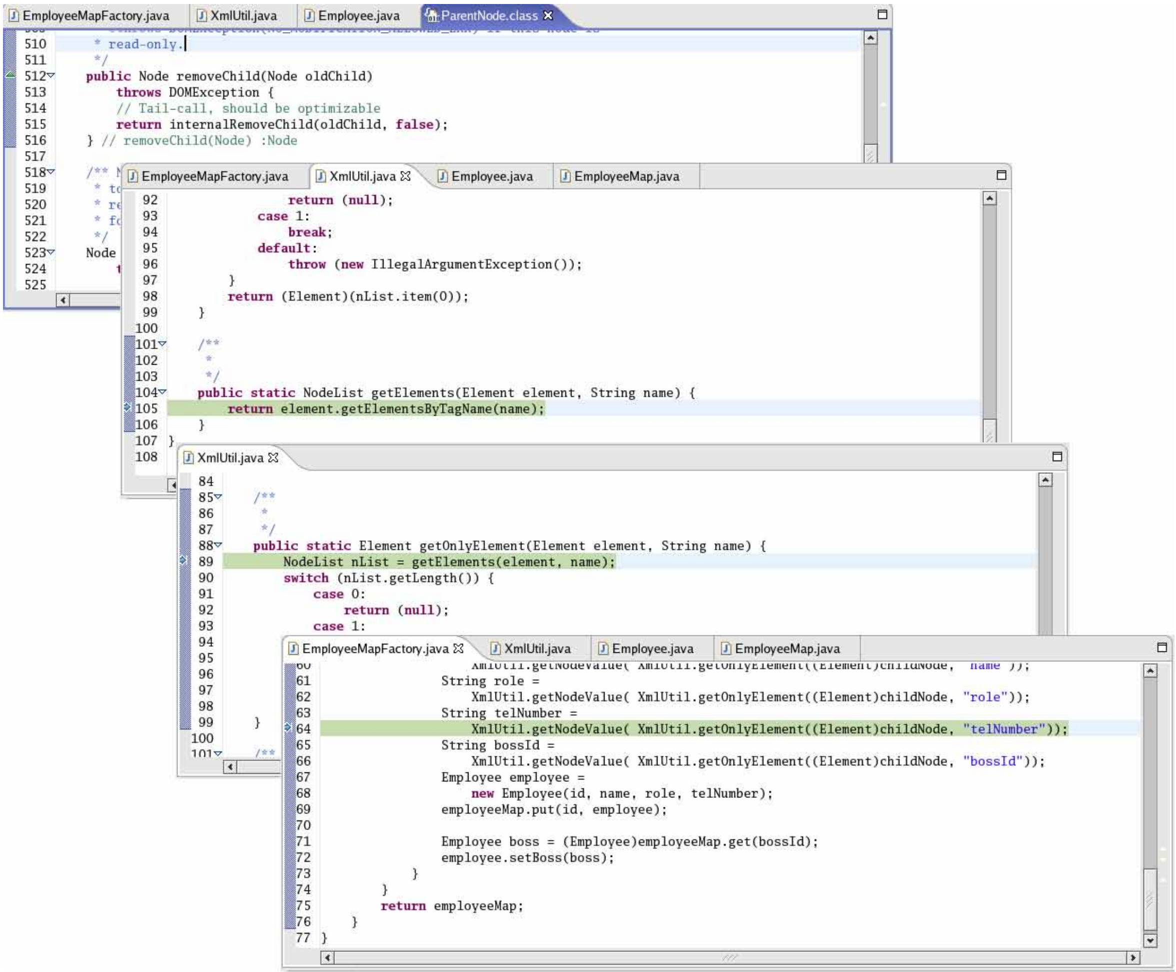


그림 5-14. [step-in]에 의한 실행

다음으로 그림 5-14의 4번째 그림에서 [step-return]단추를 찰칵해본다. 현재 메쏘드의 나머지부분이 단번에 실행되여 메쏘드를 호출한 곳에서 중지된다(그림 5-15). 걸음실행의 중단위치를 나타내는 화살표가 EmployeeMapFactory클라스의 원천코드에 돌아올 때까지 [step-return]단추를 찰칵하여 동작을 확인하여본다.

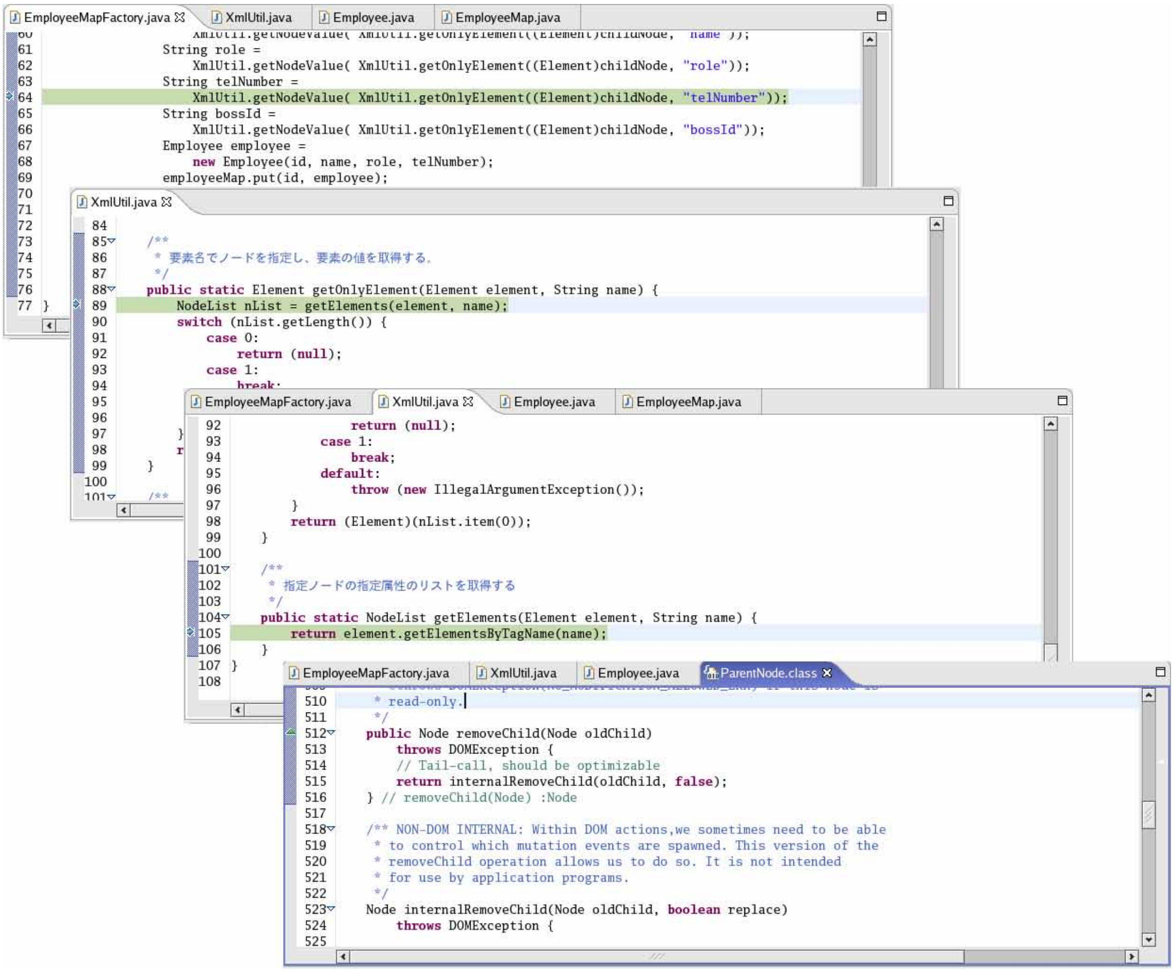


그림 5-15. [step-return]에 의한 실행

EmployeeMapFactory클라스로 돌아가서 [Resume]단추를 찰칵하여 다음의 멈춤점까지 단번에 실행해본다. 다음으로 중단하는것은 감시점으로 설정한 Employee클라스의 name마당에 대입하는 부분이다.(그림 5-16)

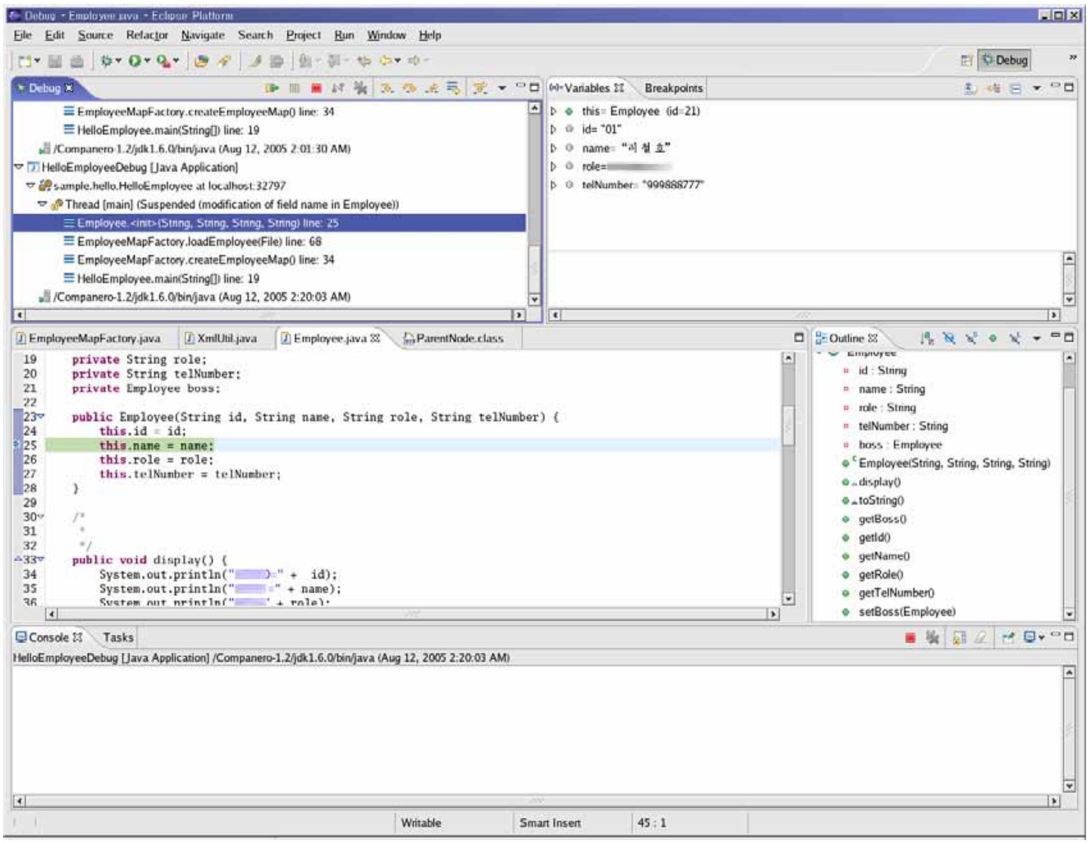


그림 5-16. 감시점을 설정한 변수의 변경에 의한 중단

5.3.3. [Variables]보기

여기서 [Debug]투시의 웃부분오른쪽에 있는 [Variables]보기를 본다(그림 5-17). 다른 보기가 현시되여있는 경우에는 [Variables]태브를 찰칵하여 [Variables]보기를 현시한다. [Variables]보기에는 멈춤점에서 참조할수 있는 변수목록이 있다. 지금은 Employee자체의 구체례의 참조인 this와 메쏘드의 인수에 넘겨지는 국부변수들인 id, name, role, telNumber가 표시되였다. ▽단추를 찰칵하여 전개하면 그 구체례에 설정되여있는 마당값을 조사할수 있다. this를 전개하면 그림 5-17과 같이 성원마당의 name에는 어떤 값도 설정되여있지 않다는것을 알수 있다.



그림 5-17. [Variables(변수)]보기에서 객체의 마당을 현시

[step-over]단추를 찰칵해본다. [Variables]보기를 보면 Employee의 name마당에 《리철호》라고 설정되여있는것을 볼수 있다(그림 5-18).

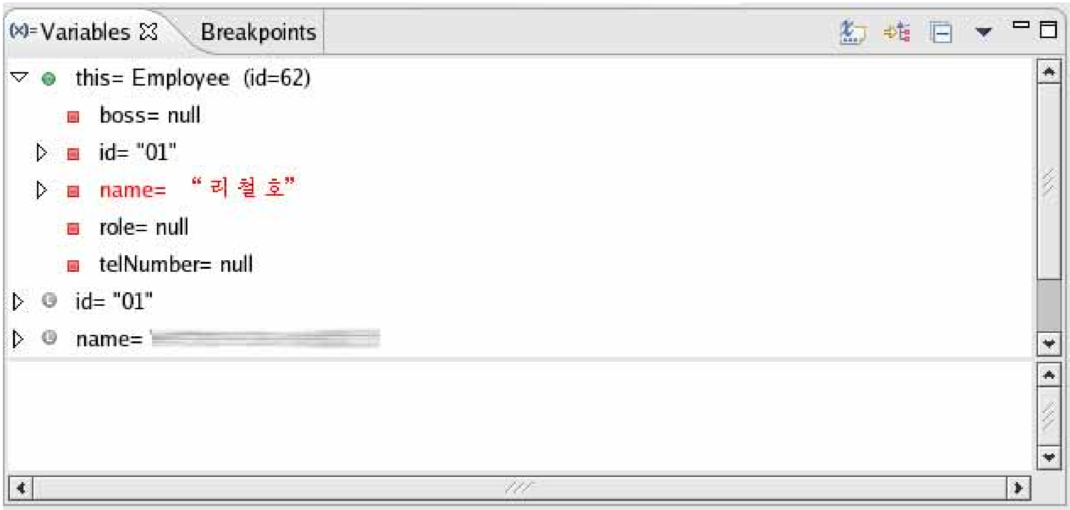


그림 5-18. name마당에 값을 설정

[Variables]보기에서는 변수의 값을 변경할수도 있다. [Variables]보기에서 this의 name마당을 두번 찰칵한다. 그림 5-19에서 보여준 대화칸이 현시되는데 《리금성》으로 변경시키고 [OK]단추를 찰칵한다.

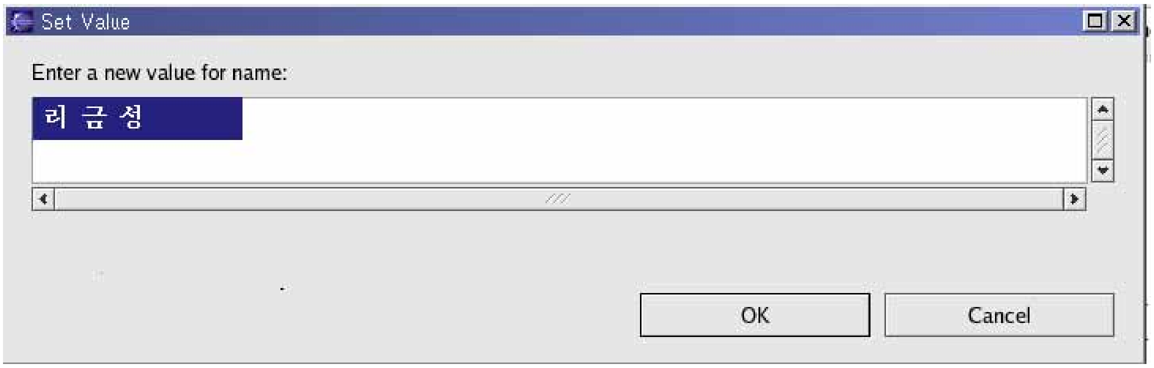


그림 5-19. 값설정대화칸

[Variables]보기에서 값이 변경된 곳을 확인할수 있다(그림 5-20).

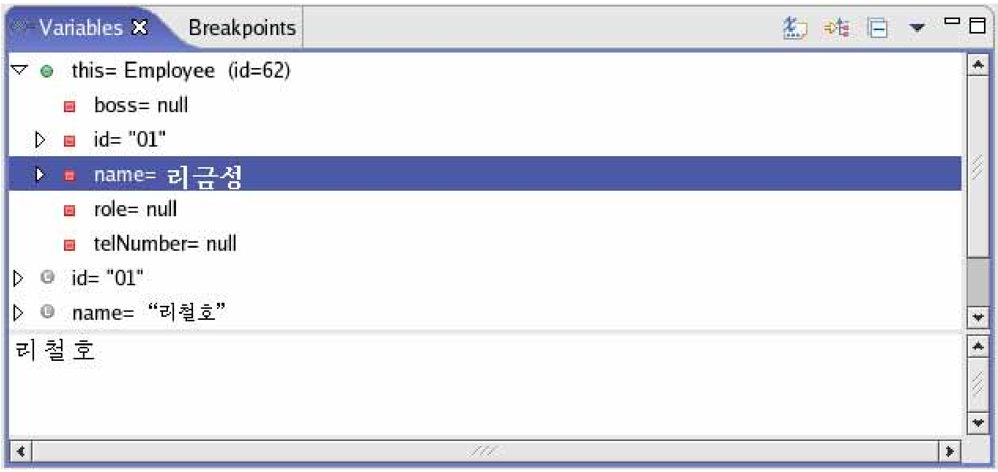


그림 5-20. name마당의 값 변경

다시 [Resume]단추를 찰칵한다. Employee클라스의 다음 구체레가 생성되여 name마당에 값이 설정되기때문에 감시점의 멈춤점에 의해 실행이 중단되는것을 확인할수 있다(그림 5-21). 또 한번 [Resume]단추를 찰칵한다. 세번째 구체레가 생성될 때 다시 중단되기때문에 다시 [Resume]단추를 찰칵한다.

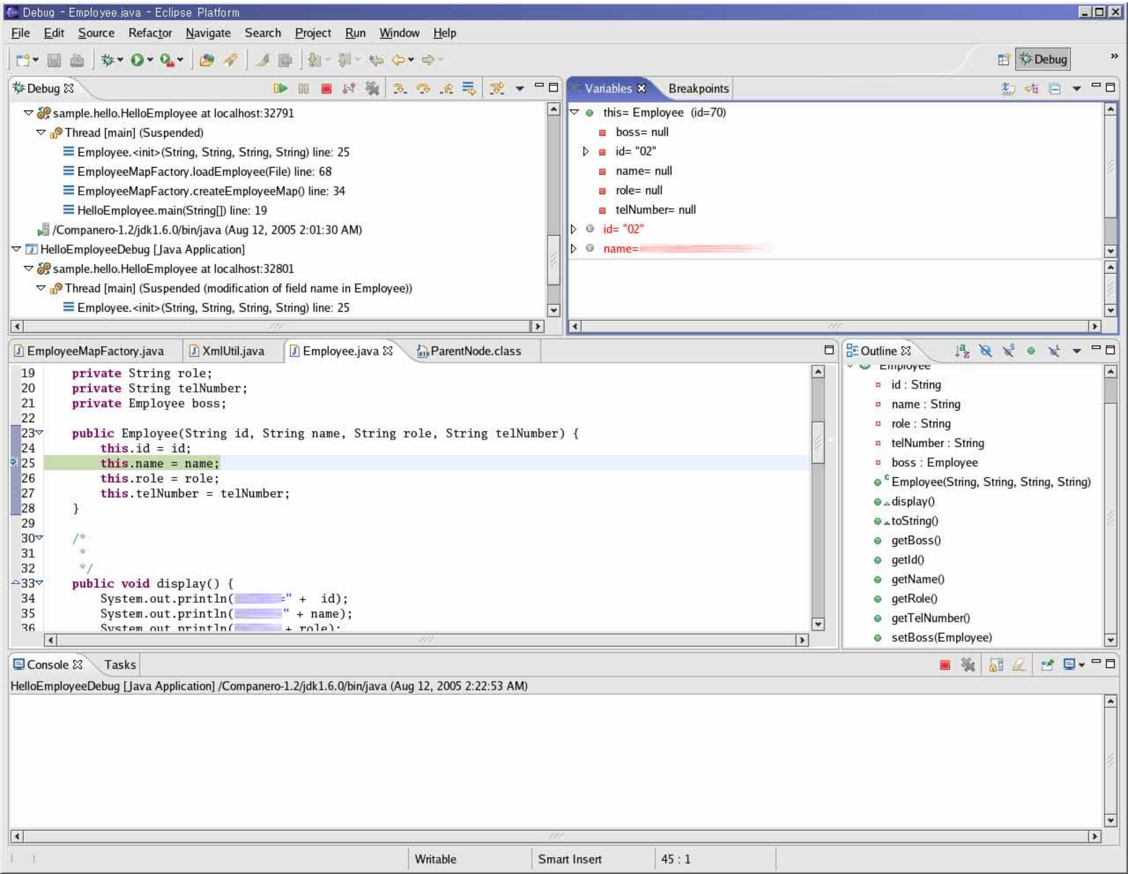
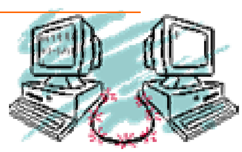


그림 5-21. 다른 구체례의 감시점에서의 중단

다음번에 중단하는것은 그림 5-22와 같이 Employee.Java의 display메쏘드이다. 여기서 감시점이 설정되여있는 name마당을 참조하고있기때문에 [outline]보기에서 name마당을 찰칵하여 name의 선언부를 표시하고 감시점표식을 오른쪽찰칵하여 [Disable Breakpoint]를 선택한다. 이렇게 실행이 중단된 상태에서도 멈춤점의 《설정/제거》나 《사용가능/사용불가능》의 변경을 진행할수 있다.

[Resume]단추를 찰칵하면 감시점을 무효로 하기때문에 다른 Employee클라스의 구체례의 display메쏘드에서 중단하는것이 아니라 제일 마지막까지 실행되여 [console]보기에 문자렬이 출력된다. [console]보기에 출력된 문자렬이 XML파일에서 읽어들인 값으로부터 변수마당에서 변경된 값으로 되였는가를 확인한다(그림 5-23).



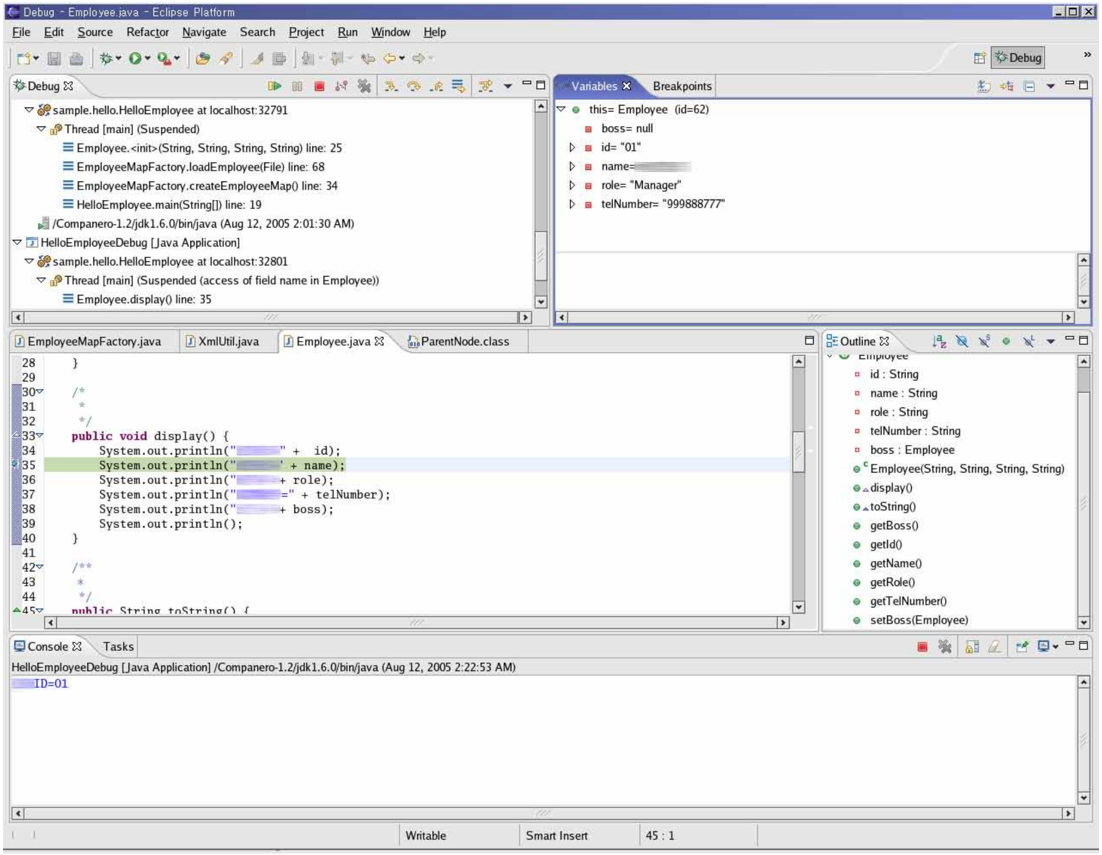


그림 5-22. 감시점을 설정한 변수의 참조에 의한 중단

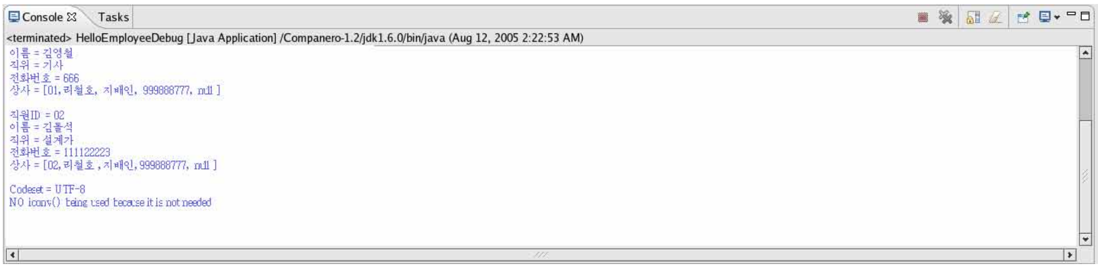


그림 5-23. [Console]보기의 출력결과

5.3.4. [BreakPoints]보기

[Debug]투시의 오른쪽우에 있는 [BreakPoints]태브를 찰칵하여 [BreakPoints]보기를 현시한다(그림 5-24). [BreakPoints]보기에는 《사용가능/사용불가능》에 관계없이 현재 설정되여있는 멈춤점의 목록이 현시된다.

[BreakPoints]보기의 검사칸들을 선택하여 멈춤점의 《사용가능/사용불가능》을 간단히 절환할수 있다. [Skip all Breakpoints]단추를 리용하면 현재의 설정을 유지한 상태로 일시적으로 모든 멈춤점을 사용불가능하게 할수 있다. 또한 멈춤점을 오른쪽찰칵하여 표시되는 지름차림표에서 멈춤점의 제거나 속성의 변경도 할수 있다. 여기서 EmployeeMapFactory(File)메쏘드에 설정되여있는 멈춤점외의 다른 멈춤점들을 사용불가능으로 한다. 이와 같이 [BreakPoints]보기를 사용하면 멈춤점의 속성을 쉽게 변경할수 있다.

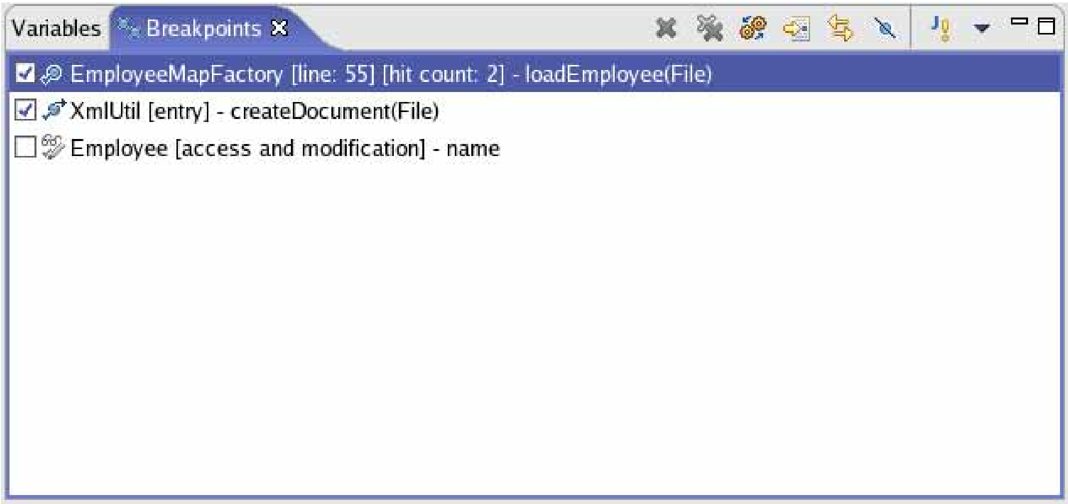


그림 5-24. [BreakPoints]보기

5.3.5. 걸음려과기(Step filter)의 사용

[5.3.2. 걸음바깥, 걸음안, 걸음되돌림]의 걸음실행에서는 [Step-in]단추를 사용하면 JRE의 체계서고에 포함되여있는 클라스의 메쏘드내부에서도 걸음실행할수 있다. 그러나 보통 JRE나 외부jar의 클라스까지 걸음실행을 하지 않는 경우가 많다. 그런 경우 자체로 작성한 클라스의 메쏘드내에서만 [step-in]동작을 하고 그 밖의 클라스에서는 [step-over]에 의하여 동작하는것이 편리하다. 그것을 가능하게 하는것이 [Debug]보기의 [Step filter/Use Step Filter]단추이다.

이 기능을 사용하기 위해서는 걸음실행을 하지 않는 클라스를 설정해 놓아야 한다. 차림표띠에서 [Window]→[Preference]을 선택하여 대화칸을 열고 대화칸의 왼쪽에서 [Java]→[Debug]→[Step Filtering]를 선택한다(그림 5-25).

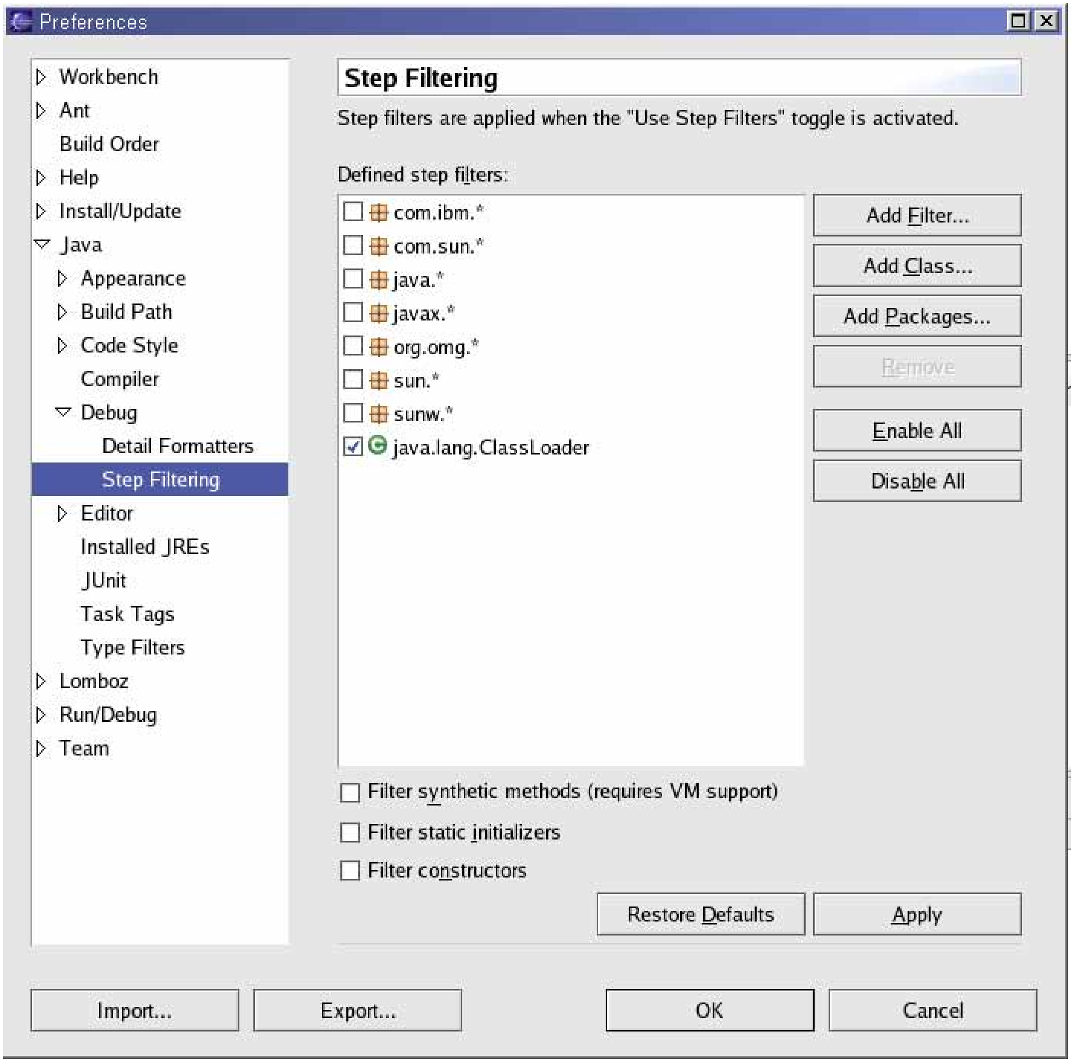


그림 5-25. 걸음려과기설정대화칸

기정으로는 여러개의 려과기가 등록되여있고 java.lang.ClassLoader려과기만이 유효하게 되여있다. 이 대화칸에서 려과기를 등록하여 유효하게 해놓으면 [Debug]보기의 [Step filter/Use Step Filter]단추가 유효하게 되며 려과기에 맞는 클라스의 메쏘드내에서 걸음실행을 하지 않게 된다.

5.3.2의 [Step-in]실행에서는 org.apache.crimson.tree.ParentNode클라스까지 들어가기때문에 이 클라스에서 걸음실행하지 않게 려과기를 등록한다. 그림 5-25의 대화칸에서 [Add Packages...]단추를 찰칵하면 그림 5-26의 대화칸이 열린다. 우의 본문입력구역에 《org.apach》라고 입력하면 아래구역에 org.apache패키지가 현시되는데

이것을 선택하여 [OK]단추를 찰칵한다. 걸음려과기의 설정대화칸에 돌아가보면 org.apache.*가 목록으로 표시되는것을 확인할수 있으며 [Enable All]단추를 찰칵한다. 등록된 파일 전체가 유효하게 된것을 확인하고 [OK]단추를 찰칵하여 대화칸을 연다.

차림표띠에서 [Run]→[debug history]→[HelloEmployeeDebug]를 선택하여 오유수정프로그람을 기동한다. 멈춤점에서 실행이 중단되면 5.3.2에서 [Step-in]실행을 개시한 장소까지 코드를 실행시킨다. 전번에는 [Step-over]단추를 사용하였지만 이번에는 편집기에서 [String role=]의 행에 지적자를 이동시키고 오른쪽찰칵하여 [Run to Line(지정한 행까지 실행)]을 선택한다. 그러면 지적자행의 앞까지의 코드가 단번에 실행되여 그 행에서 실행이 중단된다. 이 기능을 사용하면 걸음실행할 필요가 없는 도중의 코드를 단번에 실행시킬수 있으므로 기억해 둘 필요가 있다.

이렇게 5.3.2에서 [Step-in]실행을 한 위치에 도달한다. 여기서 [Step-filter/Use Step Filter]단추를 찰칵하여 이 기능을 유효(단추가 눌린 상태)상태로 한다. 이 상태에서 [Debug]보기의 [Step-in]단추를 사용하여 걸음실행을 진행하기로 한다.

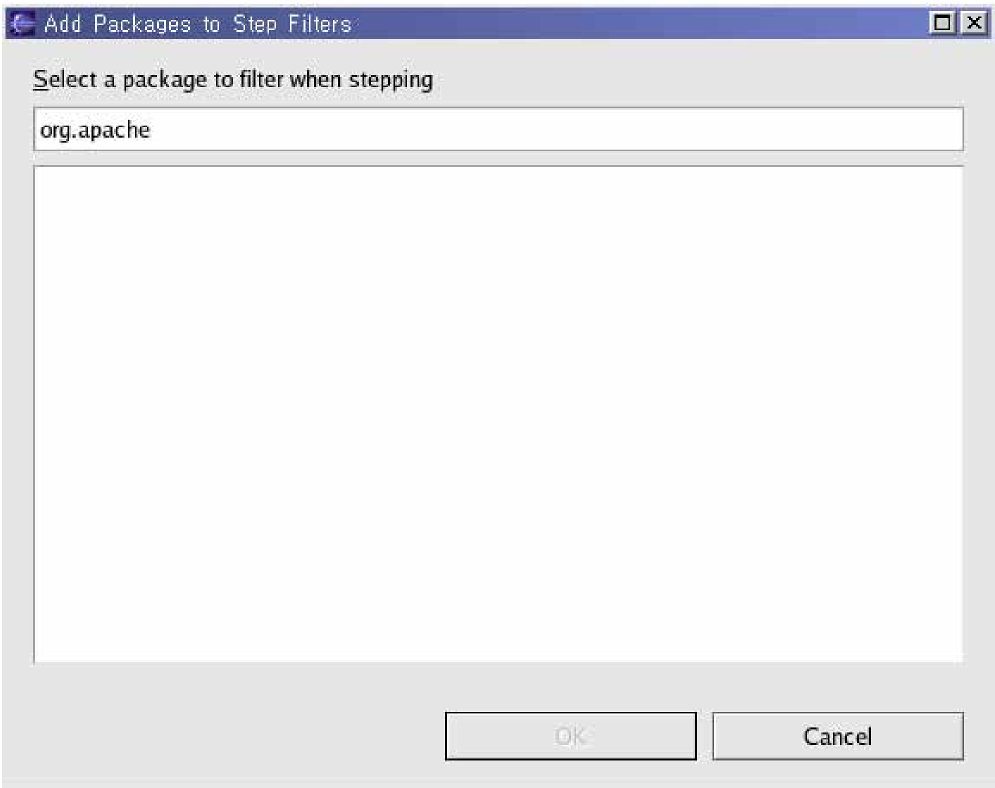


그림 5-26. 걸음려과기의 추가

5.3.2의 [Step-in]실행에서는 ParentNode클라스내부까지 걸음실행하였지만 이번에는 ParentNode클라스의 메쏘드내에 들어가지 않고 호출한 쪽으로 돌아가는것을 확인하여본다. 확인한 다음 [Debug]보기의 [Finish]단추로 오유수정프로그람의 실행을 끝마치고 려과기설정을 기정으로 되돌려 놓는다. 그리고 차림표띠에서 [Window]→[Preference]을 선택하여 대화칸을 열고 대화칸의 왼쪽에서 [Java]→[Debug]→[Step-filter]를 선택한다. 걸음려과기설정에서 오른쪽아래에 있는 [Restore Default]단추를 찰칵하면 설정이 원래대로 돌아가며 [OK]단추를 찰칵하여 대화칸을 닫는다.

[걸음려과기/걸음오유수정의 사용]의 보충

엄밀하게 [걸음려과기/걸음오유수정의 사용]이 유효로 될 때의 걸음실행은 려과기가 유효하게 된 클라스의 메쏘드만이 되는것은 아니다. 다음의 실례를 보기로 하자.

```
Public class A{
        B b=new B();
        Public void  methodA() {
                b.methodB();
        }
}
//클라스 B만이 려과기에 등록된것으로 한다.
Public class  B{
        C c=new C();
        Public void  methodB() {
                c.methodC();
        }
}
Public class C{
        Public void  methodC() {
                System.out.println(“이곳은 걸음안이다.”);
        }
}
```

여기서 클라스B가 려과기의 대상으로 되는것으로 한다. [걸음려과기/걸음오유수정의 사용]을 유효로 하고 클라스A의 methodA메쏘드를 [step-in]에서 실행시키면 b.methodB()를 실행하고 다른 클라스의 메쏘드를 걸음실행하지 않으며 methodA메쏘드가 완료되는것이 아니라 클라스C의 methodC메쏘드의 제일 첫행에서 중단된다. 결국 려과기가 유효한 클라스로부터 려과대상밖의 클라스의 메쏘드를 호출하는 경우에도 려과대상밖의 클라스의 메쏘드내부에는 [step-in]실행한다. 따라서 JRE나 외부jar파일의 클라스의 메쏘드에서 걸음실행을 하지 않도록 하려면 그것들에 포함되는 전체클라스가 대상으로 되도록 려과기를 설정해둘 필요가 있다.

5.3.6. [Display]보기, [Expression]보기

[Variables(변수)]보기를 사용하면 변수의 값을 조사할수 있지만 오유수정과정에 메쏘드의 되돌이값을 알아야 할 경우도 있다. Eclipse에서는 오유수정프로그람을 사용하여 멈춤점에서 중단하였을 때에 임의의 식을 평가할수 있다.

그 방법을 설명하기 전에 몇가지 준비가 필요하다. [Java]투시로 절환하여 [Package Explorer]보기에서 [SampleChap5]프로젝트→[src]서류철→[sample.hello]패키지→[EmployeeMap.java]를 두번 찰칵하여 원천코드를 현시한다. [Window]보기에서 display메쏘드를 선택하고 display메쏘드내의 《eArray[i].display();》에 행멈춤점을 설정한다. 다시 오유수정투시로 절환하여 [BreakPoints]보기([5.3.4. [BreakPoints]보기]를 참조)에서 이자 설정한 멈춤점외의 다른 멈춤점들을 사용불가능으로 한다. 이것으로서 준비가 완료된다.

차림표띠에서 [Window]→[Display view]→[Display]를 선택하여 [Display]보기를 현시한다(그림 5-27). [Display]보기는 [Debug]투시의 아래단에 현시된다. 차림표띠에서 [Run]→[Debug History]→[HelloEmployeeDebug]를 선택하여 오유수정프로그람을 실행시킨다. EmployeeMap클라스의 display메쏘드의 멈춤점에서 [Display]보기에 다음의 식을 입력한다.

eArray[i].getName()

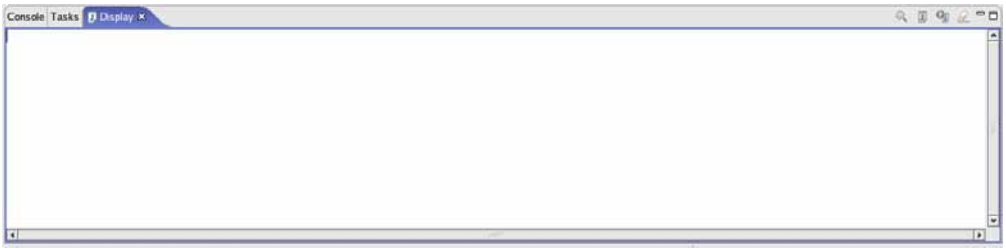


그림 5-27. [Display]보기

입력된 식을 선택하고 마우스오른쪽단추를 찰칵하여 [Display]를 선택하면 식의 평가결과가 [Display]보기에 현시된다(그림 5-28).



그림 5-28. 식의 평가결과를 [Display]보기에서 현시

다음으로 [Debug]보기에서 HelloEmployee.main(String[])의 탄창프레임을 선택하고 같은 방법으로 [Display]보기를 사용하여 앞의 식의 평가결과를 현시하여본다.

이렇게 하면 그림 5-29와 같이 식평가가 실패한다. 이것은 HelloEmployee.main(String[])의 문맥에서는 earray나 i라는 변수가 유효범위밖에 있기때문이다. 이와 같이 [Display]보기에서의 식평가는 [Debug]보기에서 선택되여있는 탄창프레임에 의존한다.



그림 5-29. 유효범위밖에서 식을 평가한 경우



그림 5-30. [Display]보기에서 [inspection]를 실행

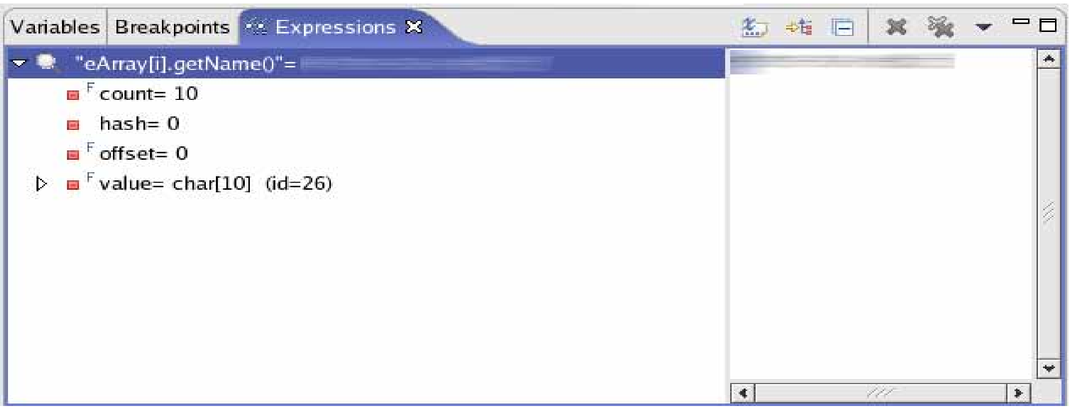


그림 5-31. [Expressions]보기에서 식평가결과를 현시

[Debug]보기에서 EmployeeMap.display()를 선택하고 원래의 탄창프레임으로 돌아간다. [Display]보기에서 앞에서 입력한 식을 선택하고 마우스오른쪽단추를 찰칵하여 [inspection]를 선택한다. [Display]를 선택하였을 때에는 되돌이값형과 값만이 현시되지만 이번에는 그림 5-30과 같이 보다 자세한 내용이 튀여나오기창에 현시된다. 여기서 ctrl+shift+l을 누르면 그림 5-31의 [Expressions]보기가 [Debug]투시의 오른쪽

우에 현시된다. [Expressions]보기에는 앞서 본 튀여나오기창의 내용이 현시되여있다.

[Debug]보기의 [Resume]단추를 찰칵하면 두번째 Employee클라스의 구체례의 정보를 조종탁에 출력하기직전에 중단된다. 이때 eArray[i].getName()의 평가결과는 [김돌석]이 아니지만 [Expressions]보기에서는 변화되지 않는다. 이와 같이 [inspection]의 경우 식평가는 한번뿐이다. 식을 계속하여 평가하려면 [Expressions]보기에서 [eArray[i].getName()]을 선택하여 마우스오른쪽단추를 찰칵하고 [Reevaluate Watch Expression]을 선택한다. 그림 5-32와 같이 현시가 바뀌여 방금 중단된 위치에서의 평가식이 현시된다. [Debug]보기에서 [Resume]단추를 찰칵하여 다음 멈춤점에서 중단되였을 때에는 식을 평가하여 고쳐진것을 확인한다.

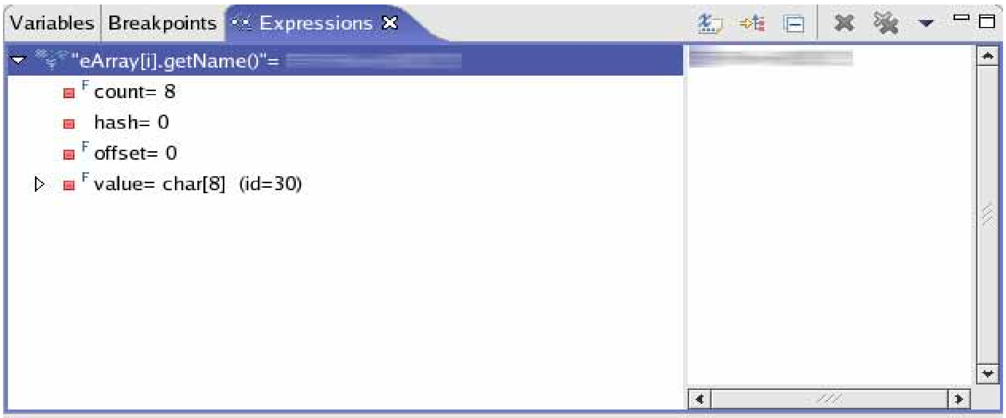


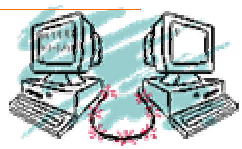
그림 5-32. [Reevaluate Watch Expression]실행후 결과

제4절. 오유수정의 실레

여기서는 Junit시험이 실패한 레를 리용하여 실제로 오유수정를 진행해본다. 앞절들에서 설정한 멈춤점은 [BreakPoints]보기 등을 사용하여 모두 제거한다. [Java]투시의 [Package Explorer]보기에서 SampleChap5프로젝트를 연다. [test]서류철→[sample.hello]패키지→[EmployeeMapFactoryTest.java]의 원천코드를 현시하고 testGreatEmployeeMap메쏘드의 첫 행을 목록 5-1과 같이 employee.xml로부터 employee2.xml로 변경시킨다.

목록 5-1. EmployeeMapFactoryTest#testCreateEmployeeMap() (일부)

```
public   void   testCreatEmployeeMap() throws Exception {
    {EmployeeMapFactory eFactory=new EmployeeMapFactory( “employee2.xml” );
        EmployeeMap eMap =eFactory.createEmployeeMap();
```

목록 5-2. Employee2.xml

```
<?xml version= '1.0'  encoding= "shift_JIS" ?>
<EmployeeList>
  <Employee id  = "02" >
        <name>김영수</name>
        <role>designer</role>
        <telNumber>111122223</ telNumber>
        <bossId>01</bossId>
  </Employee>
  <Employee id  = "03" >
        <name>김돌석</name>
        <role>Engineer</role>
        <telNumber>666</ telNumber>
        <bossId>01</bossId>
  </Employee>
  <Employee id  = "01" >
        <name>리철호</name>
        <role>manager</role>
        <telNumber>999888777</ telNumber>
        <bossId>
  </Employee>
 </EmployeeList>
```

변경시킨 시험경우를 실행시켜 보기로 한다. [Package Explorer]보기에서 EmployeeMapFactoryTest.java를 선택한 상태에서 차림표띠에서 [Run]→[Run As]→[Junit test]을 선택한다. 시험은 실패하고 [Junit]보기에 실패한 시험의 정보가 현시된다(그림 5-33). 오유추적에서 제일 꼭대기의 탄창프레임을 두번 찰칵하면 편집기에 실패한 assertEquals문이 현시된다. [Junit]보기의 통보문에서 알수 있는바와 같이 e02.toString()의 결과가 예측한 문자렬과 다른것이다.

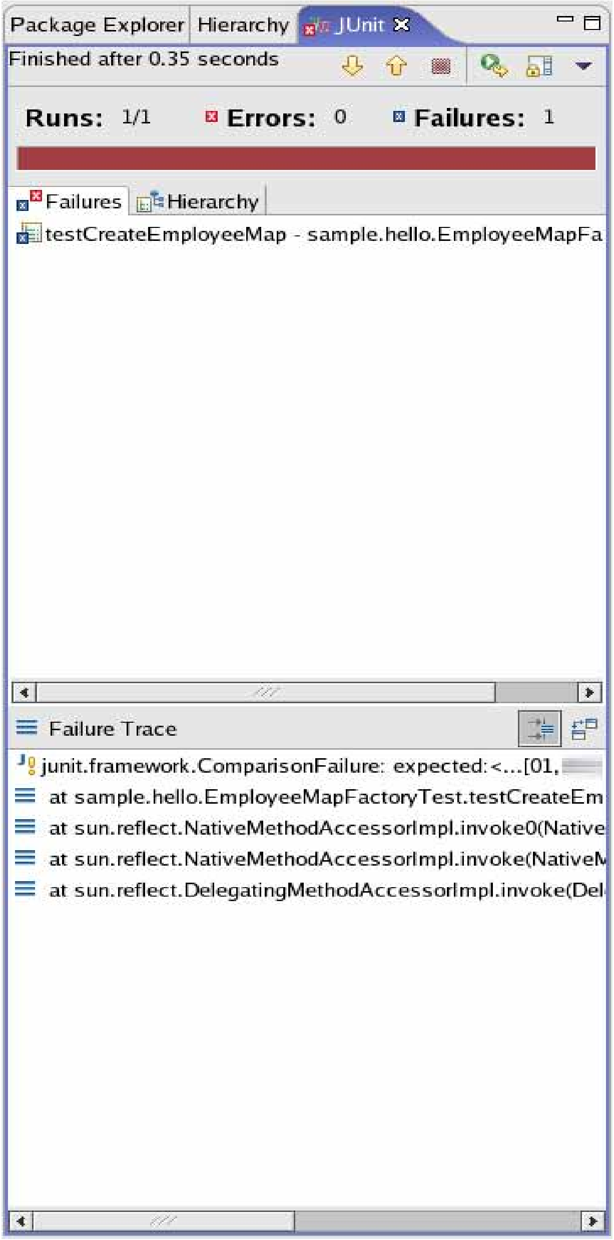


그림 5-33. 실패한 검사

좀더 자세히 보기로 한다. 앞에서 실패한 assertEquals문에 행멈춤점을 설정한다.

멈춤점을 설정한 다음 차림표띠에서 [Run]→[Debug History]→[EmployeeMapFactoryTest]를 선택하여 오유수정프로그람을 기동한다. 멈춤점에서 실행이 중단되므로 e02.toString()의 값을 알수 있다. Java편집기에서 e02.toString()을 마우스로 끌기하여 반전시킨다(그림 5-34).

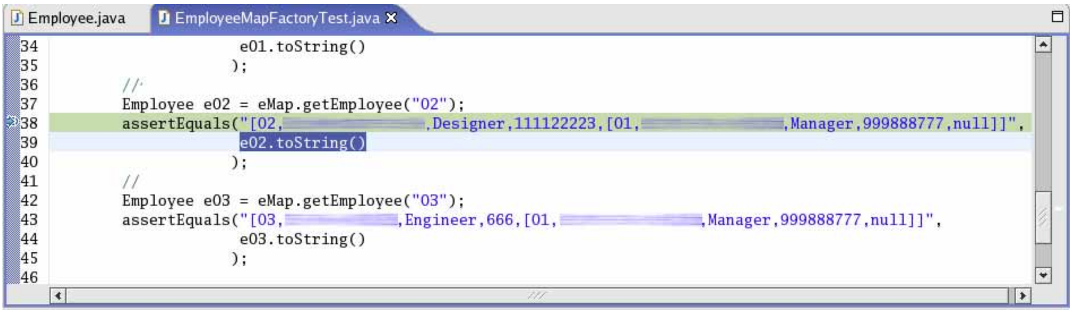


그림 5-34. e02.toString()의 되돌림값을 조사

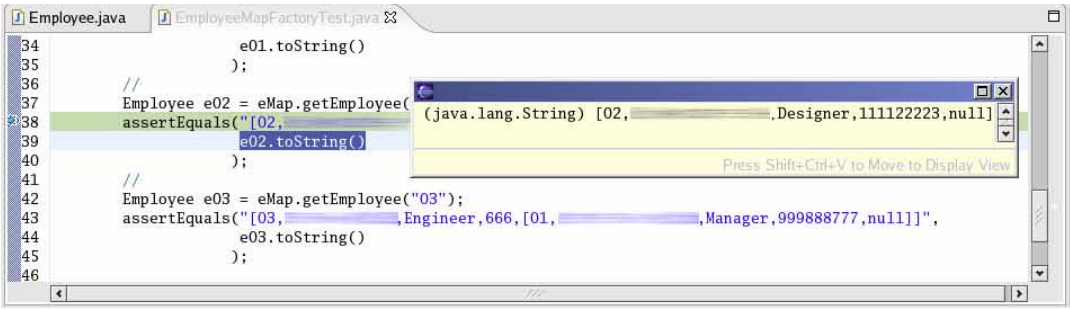


그림 5-35. e02.toString()의 평가결과

assertEquals문에서 문자렬을 비교하면 《 01,리철호,공개원천프로그람**,Manager.999888777,null》부분이 null로 된다(그림 5-35). [Variables]보기를 현시하여 e02마당을 조사하면 boss마당이 null로 되는것이 문제이다(그림 5-36).

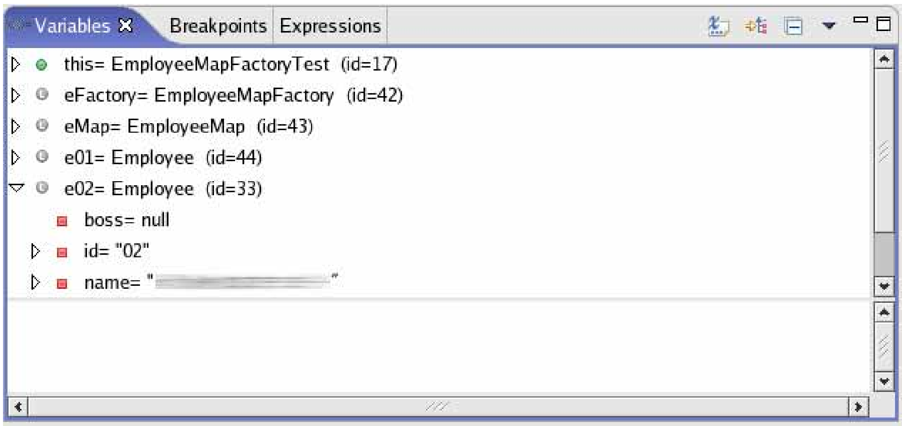


그림 5-36. e02마당의 값

e02의 boss마당이 id= "01" 인 Employee객체를 참조하고있다면 toString()의 결과가 예상한대로 되는가를 확인한다. [Variables]보기를 보면 e01에 e02의 boss마당에 설정할수 있는 객체가 설정되여있다. [Display]보기를 사용하여 e02.setBoss(e01)의 코드토막을 실행하여(그림 5-37) 다시 한번 java편집기에서 e02.toString()을 선택→오른쪽찰칵→[Display]에서 결과를 확인한다(그림 5-38). 이렇게 하면 assertEqual문에서 비교하려는 문자렬과 같은 문자렬로 되므로 Boss마당이 정확히 설정되지 않았다는 가정이 옳게 된다.

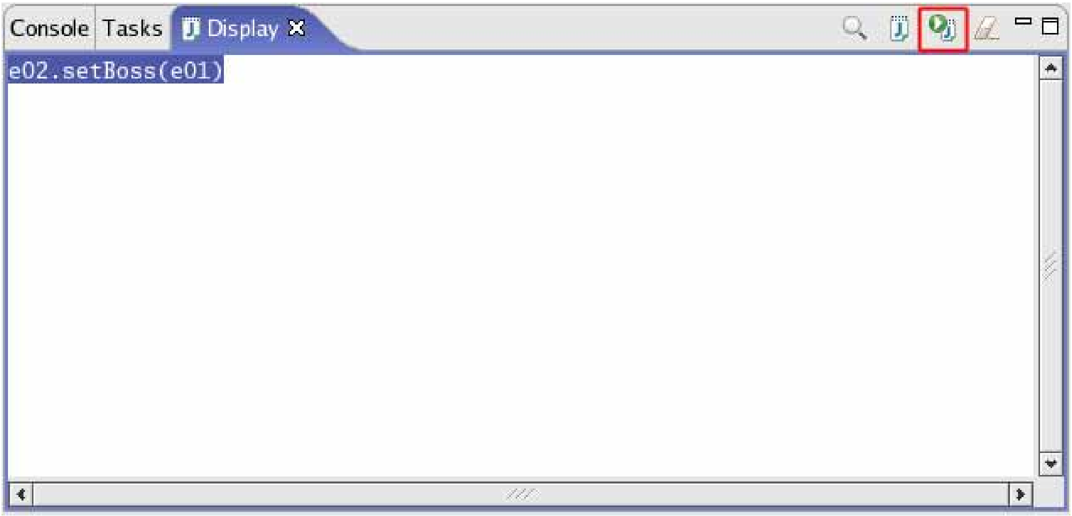


그림 5-37. e02.setBoss(e01)을 실행

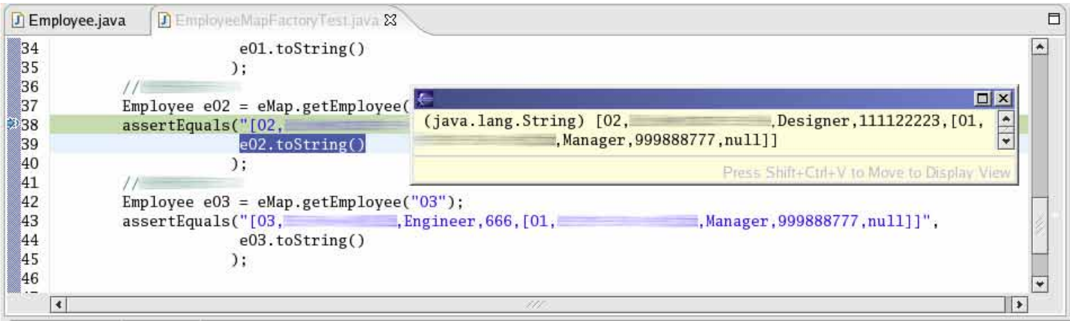


그림 5-38. setBoss실행후의 e02.toString()의 평가결과

여기서 boss마당이 변경되였을 때의 변수값을 조사해보기로 한다. [Debug]보기의 [Finish]단추를 찰칵하여 일단 실행을 중지시킨다. [Java]투시로 절환하여 Employee.java의 원천코드를 현시하고 boss마당에 감시점을 설정한다. 실행시에

boss마당에 설정되여있는 객체를 확인하려면 멈춤점의 속성에서 [field access]의 선택을 제거하여 놓는다.

[Debug]투시로 절환하여 차림표띠로부터 [Run]→[Debug History]→[EmployeeMapFactoryTest]를 선택하여 다시 한번 오유수정프로그람을 기동한다. 이렇게하면 Employee클라스의 setBoss메쏘드에서 실행이 중단된다.

[Debug]보기에서 현재 탄창프레임의 바로밑에 있는 EmployeeMapFactory.loadEmployee(file)의 탄창프레임을 선택한다. [Variables]보기에서 변수 employee를 보면 id="02"의Employee클라스의 구체례를 참조하고있다. setBoss(boss)메쏘드의 인수에 넘겨진 국부변수 boss를 [Variables]보기에서 조사하면 null로 되여있다(그림 5-39).

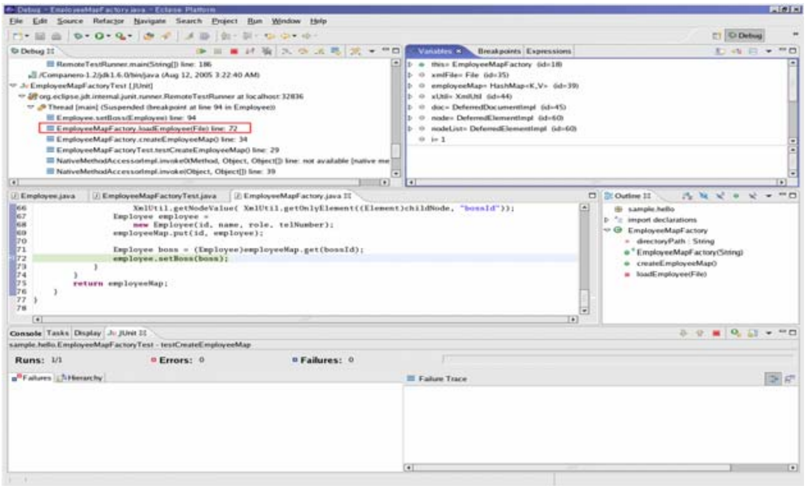


그림 5-39. 탄창프레임의 선택

국부변수 boss에는 화살표로 지적된 코드의 한 행앞에서 employeeMap.get(bossId)의 결과가 설정되여있다. [Variables]보기를 보면 bossId는 01로 되여있으므로 id= "01" 인 Employee객체를 얻게 된다. 그래서 [Variables]보기로 EmployeeMap의 상태를 조사한다. [Variables]보기에서 [employeeMap]의 왼쪽에 있는 ▷단추를 찰칵하여 employeeMap의 내용을 현시한다. 여기서 [Show Logical Structure]아이콘을 찰칵하여 현시를 절환시켜 본다(그림 5-40). [론리구조]가 유효로 되면(눌리여진 상태) HashMap나 ArrayList 등의 집합클라스가 가지고있는 객체만이 현시된다. 분명 employeeMap에는 id= "02" 인 Employee객체밖에 설정되지 않고 id= "01" 의 Employee객체는 존재하지 않는다.

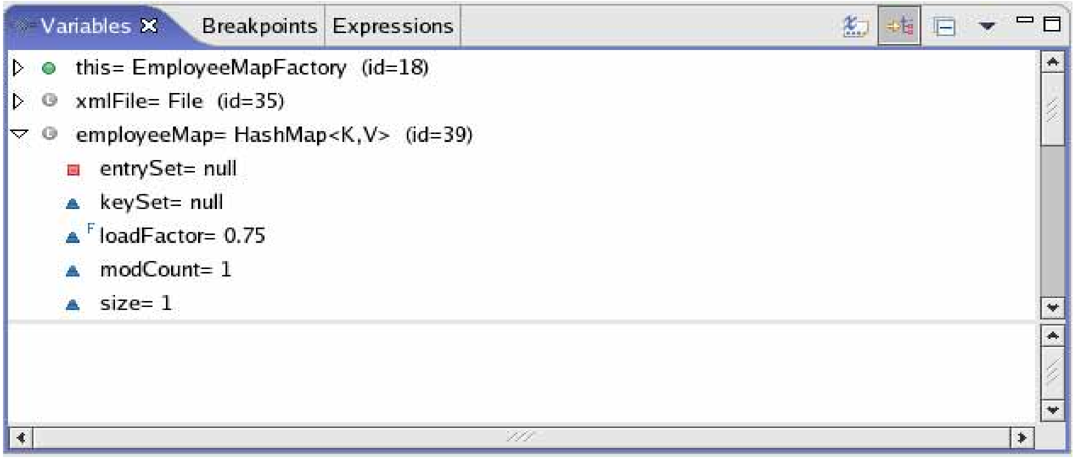


그림 5-40. employeemap의 상태

왜 id= “01”의 Employee객체가 존재하지 않게 되는가? Employee객체는 employee2.xml파일을 기초로 생성되지만 생성되는 순서는 employee2.xml파일에서 서술된 순서로 된다. 따라서 id= “01”의 Employee의 객체는 id= “02”의 객체보다도 후에 생성되기때문에 이 시점에서는 id= “01”의 객체를 참조할수가 없다. 결국 상급(boss)에 해당한 Employee객체가 먼저 생성된 경우에는 문제가 없겠지만 상급의 Employee객체가 하급의 객체보다도 후에 생성되는 경우에 하급으로부터 상급에로의 관련설정이 정확히 설정되지 않는 오유가 이 프로그람에 존재한다.

이것을 해결하는 방법에는 여러가지가 있으나 여기에서는 모든 Employee객체가 생성된 다음에 상급과 하급의 관련을 설정하는 방법에 대하여 취급한다. 판본1.4이후의 Java가상기계를 사용하는 경우는 멈춤점에서 중단된 상태에서 코드를 수정하여 계속 실행시킬수 있다. EmployeeMapFactory클라스의 loadEmployee메쏘드를 목록 5-3과 같이 수정한다.

목록 5-3. EmployeeMapFactory#loadEmployee(file)

```
/**
*파일에서 직원정보를 읽기한다.
*@param employeeMap
*@param xmlFile
*/
private java.util.map loadEmployee(file xmlfile){
  java.util.map employeemap = new hashMap();
  //(5.4)id와 bossId의 대조를 일시보관하기 위한 HashMap
  java.util.map bossmap=new HashMap();

  XmlUtil xUtil = null;
```

```
    Document doc=null;
    Try {
       xUtil =new XmlUtil();
       doc = xUtil.createDocument(xmlfile);
    }catch (Exception e) {
       e.printStackTrace();
    }
    Node node = doc.getDocumentElement();
    NodeList nodeList = node.getChildNodes();
    for (int i=0;  i< nodeList.getLength(); i++) {
       Node childNode = nodeList.item(i);
       if (chidNode.getNodeType()==Node.ELEMENT_NODE){
         String id=XmlUtil.getAttributeValue(childNode,  "id");
         String name=XmlUtil.getAttributeValue
            (XmlUtil.getOnlyElement((Element)childNode, "name"));
         String role=getNodeValue(XmlUtil .getOnlyElement(
            (Element)childNode,  "role"));
         String telNumber=getNodeValue(XmlUtil .getOnlyElement(
            (Element)childNode,  "telNumber"));
         String bossId=getNodeValue(XmlUtil .getOnlyElement(
            (Element)childNode,  "bossId"));
         Employee employee=new Employee(id, name, role, telNumber);
         employeeMap.put(id, employee);
         //(5.4) 다음의 2개 행에 설명문을 붙인다.
         //Employee boss = (Employee)employeeMap.get(bossId);
         // employee .setBoss(boss);
         //(5.4) bossId를 림시 보관한다.
         bossMap.put(id, bossId);
       }
    }

    for (java.util.Iterator itr=bossMap.keySet().iterator(); itr.hasNext();){
        String id=(String) itr.next();
        String bossId=(String)bossMap.get(id);
        ((Employee) employee.get(id)).setboss(
          (Employee) employeeMap.get(bossId));
    }

    return employeeMap;
  }
```

수정한 다음 편집한 원천코드를 보존한다. 이때 코드가 콤파일되여 현재 실행중인 메쏘드의 제일 첫 행으로 중단위치를 나타내는 화살표가 이동하게 된다(그림 5-41). 결국 수정된 메쏘드의 제일 처음부터 실행이 재개된다.

```
41      */
42      private java.util.Map loadEmployee(File xmlFile) {
43          java.util.Map employeeMap = new HashMap();
44          java.util.Map bossMap=new HashMap();///
45          XmlUtil xUtil = null;
46          Document doc = null;
47          try {
48              xUtil = new XmlUtil();
49              doc = xUtil.createDocument(xmlFile);
50          } catch (Exception e) {
51              e.printStackTrace();
52          }
53          Node node = doc.getDocumentElement();
54          NodeList nodeList = node.getChildNodes();
```

그림 5-41. 코드수정후의 중단위치

Hot코드치환

판본1.4이후의 Java가상기계를 사용하는 경우는 멈춤점에서 중단된 상태에서 코드를 수정하고 계속하여 수정후의 코드를 실행시킬수 있다. 이것을 **Hot코드치환**이라고 한다. 그러나 Hot코드치환에 대응하지 않는 가상기계를 사용하는 경우나 다음과 같이 클라스의 구조를 변경시키는 수정을 하는 경우에는 hot코드치환에서 실패한다.

- 메쏘드의 추가나 삭제
- 파일의 추가나 삭제
- 내부클라스의 변경(외부클라스의 private마당에 대응하는 접근의 추가나 삭제)

이 경우 실행상태에 수정된것이 반영되지 않으므로 수정후의 코드를 실행하기 위하여서는 오유수정프로그람을 기동하여 고칠 필요가 있다. Eclipse3.0부터는 hot코드치환에서 실패하였을 때 현시되는 통보문대화칸에 오유수정프로그람을 재기동시키는 단추가 추가되여있다.

[Debug]보기에서 [Resume]단추를 찰칵하여 멈춤점에서 중지되면 다시 한번 [Resume]단추를 찰칵하여 [Variables]보기에서 boss마당에 설정된 값을 관찰한다. id="03"인 Employee객체의 boss마당에 id="01"인 Employee객체가 설정되도록 하였다(그림 5-42). 걸음실행단추나 [Resume]단추를 사용하여 처리를 전진시키고 boss마당에 Employee객체가 설정되는것을 확인하면서 EmployeeMapFactoryTest를 마지막까지 실행시켜 본다. 시험의 실행이 끝나면 [Debug]투시의 제일아래에 있는 [JUnit]태브를 찰칵하여 [Junit]보기를 현시시켜 본다. 푸른 띠가 현시되면 시험이 통과된것으로 된다.

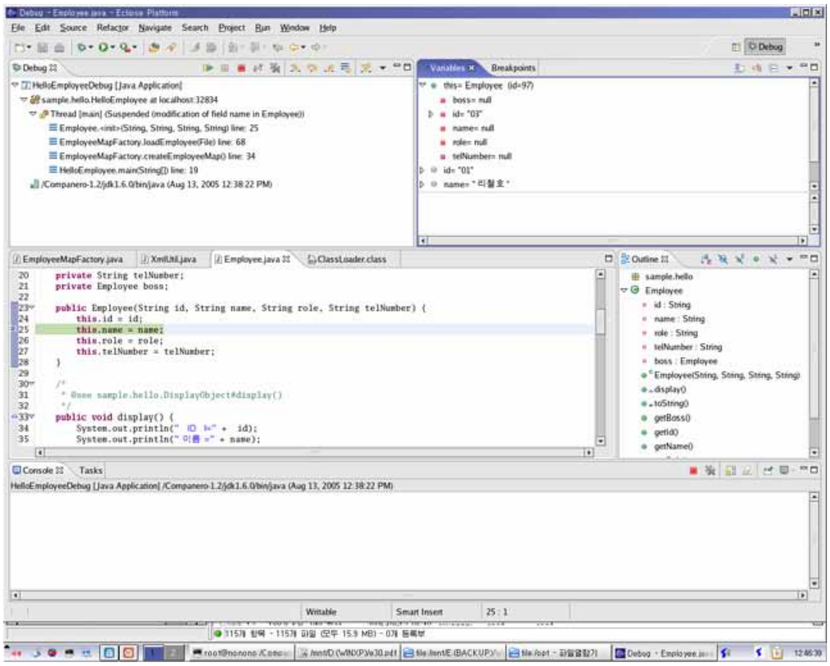


그림 5-42. 코드수정후의 실행

제6장. JDT의 고급한 사용법

제3장부터 5장까지에서 프로그람작성, 콤파일, 실행, 시험, 오유수정과 일반적인 조작들에 대하여 설명하였다. Eclipse를 리용한 Java응용프로그람의 개발방법에 대하여 기본적으로 리해하였다고 보면서 이장에서는 실제로 개발작업에 쓸모있는 JDT(Java개발도구)의 기능에 대하여 몇가지 소개한다.

제1절. [Java Browsing]투시

차림표띠에서 [Window]→[Open perspective]→[Java Browsing]를 선택하면 [Java Browsing]투시가 열린다(그림 6-1). 또한 투시절환도구의 [Open perspective]→[Java Browsing]에서도 [Java Browsing]투시를 열수 있다.

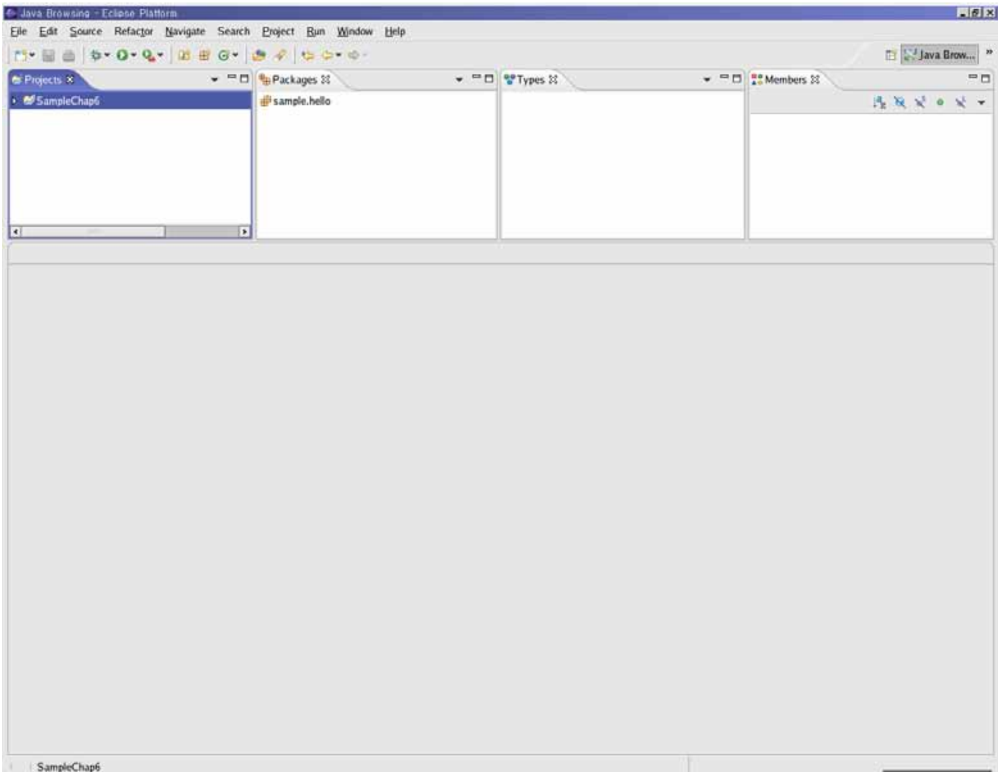


그림 6-1. [Java Browsing]투시

웃부분에 4개의 보기가 나란이 놓여있는데 왼쪽으로부터 [Projects]보기, [Packages]보기, [Types]보기, [Members]보기가 있다.

[Project]보기는 프로젝트를 현시하기 위한것인데 프로젝트를 두번찰칵하여 전개하면 그 프로젝트에 설정되여있는 건립경로(서류철 또는 jar파일)가 현시된다. [Project]보기에서 원천서류철이나 jar파일을 선택하면 그 서류철 또는 jar파일에 포함되여있는 패키지목록이 [Packages]보기에 현시된다. [Packages]보기에서 패키지를 선택하면 그 패키지에 포함된 형(클라스 또는 대면)목록이 [Types]보기에 현시된다. [Types]보기에서 형을 선택하면 형의 성원(변수 또는 메쏘드)이 [Members]보기에 현시된다.

원천코드가 편집기령역에 현시되지 않을 때에는 [Types]보기에서 형을 두번찰칵하면 선택된 형의 원천코드가 편집기령역에 표시된다. [Members]보기에서 성원을 두번찰칵하면 편집기령역에 원천코드가 열리며 선택된 성원의 선언부가 현시된다. jar파일에 포함되여있는 클라스 등에 원천코드가 첨부되여있지 않는 경우는 해당한 코드가 없다는 통보문을 내보내고 그 클라스성원의 서명(signature)일람이 편집기령역에 현시된다(그림 6-2).

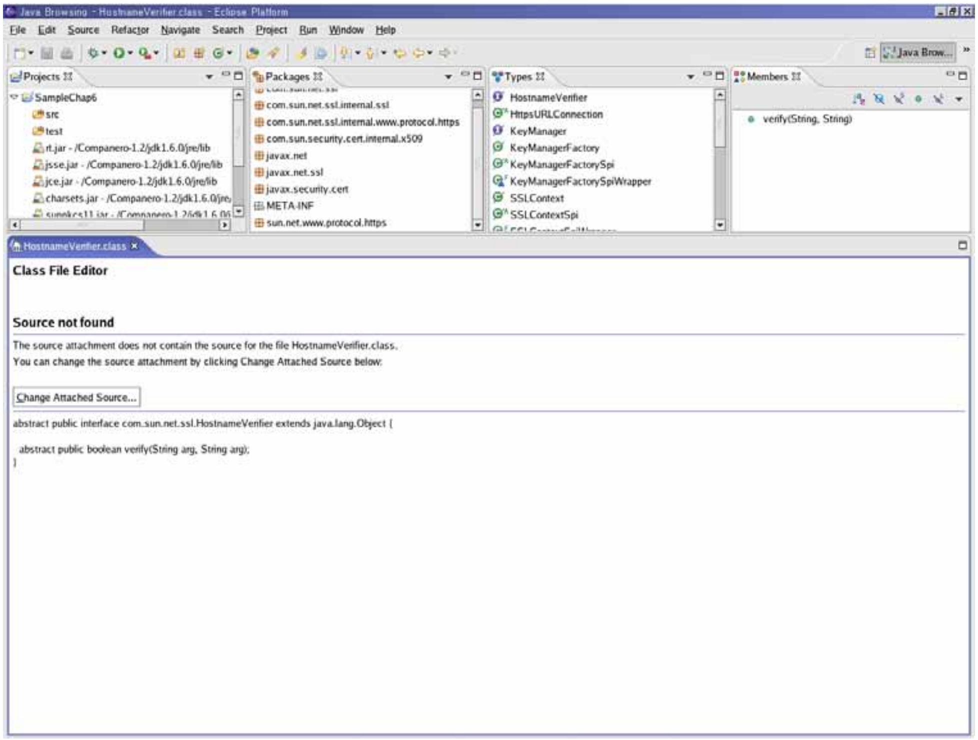


그림 6-2. 원천코드가 첨부되여있지 않는 경우

실제로 org.w3c.dom.Document대면의 성원을 현시하기로 해보자.

1. Eclipse를 기동하고 SampleChap6프로젝트를 연다.
2. 차림표띠에서 [Window]→[Open Perspective]→[Java Browsing]를 선택하여 [Java Browsing]투시를 연다.

3. [Project]보기에서 sampleChap6프로젝트를 열고 rt.jar를 선택한다.
4. [Packages]보기에서 [org.w3c.dom]을 선택하고 [Types]보기에서는[Document]를 선택하며 [Members]보기에서 [createElement(string)]을 두번찰칵한다(그림 6-3).

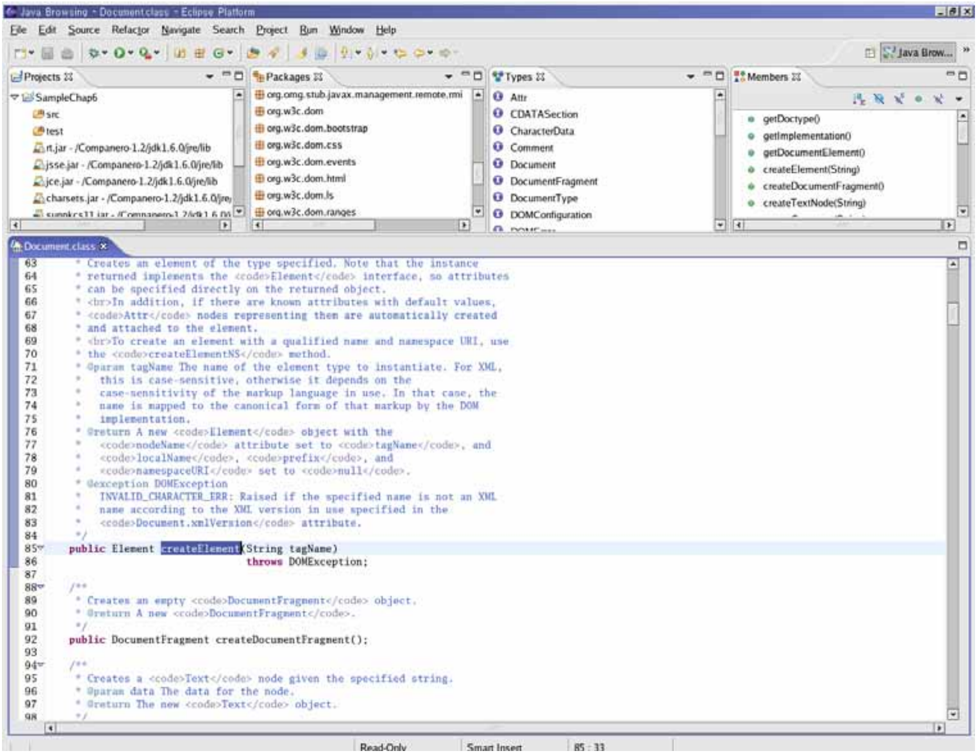


그림 6-3. org.w3c.dom.Docunent의 원천코드를 현시

[Java Browsing]투시는 기능적으로는 [Java]투시와 크게 차이가 없지만 참조에서 대상을 좁혀서 배치하였다. [Java]투시의 [Package Explorer]보기와 [Java Browsing]투시의 4개의 보기가운데서 어느것을 사용하는가 하는것은 개발자들의 선택하는데 달려있지만 보기들이 분할되여있는 [Java Browsing]투시가 참조나 조작하는데서 더 좋을것이다. 또한 [Java]투시보다 편집기령역을 넓게 사용할수 있는것도 이 투시의 우점이다.

제2절. [Hierarchy]보기

Java클라스나 대면의 계승관계를 조사할 때 편리한것이 [Hierarchy]보기이다. [Hierarchy]보기를 현시하기 위하여서는 [Package Explorer]보기에서 형(클라스나 대면)을 선택한 다음 F4건을 누른다. 형을 오른쪽찰칵→ [Open Type Hierarchy] 또는 차림표띠의 [Navigate]→[Open Type Hierarchy]에서도 [Hierarchy]보기를 열수 있다. 여기서는 [Java]투시의 [Package Explorer]보기에서 [SampleChap6]프로젝트→[JRE System Library]→[rt.jar]→[java.util]패키지→[ArrayList.class]를 선택한 다음 F4건을 눌러 그림 6-4와 같은 [Hierarchy]보기를 현시한다. 그림 6-4와 다른 경우는 [Hierarchy]보기의 [Show the Type Hierarchy]단추를 찰칵한다.

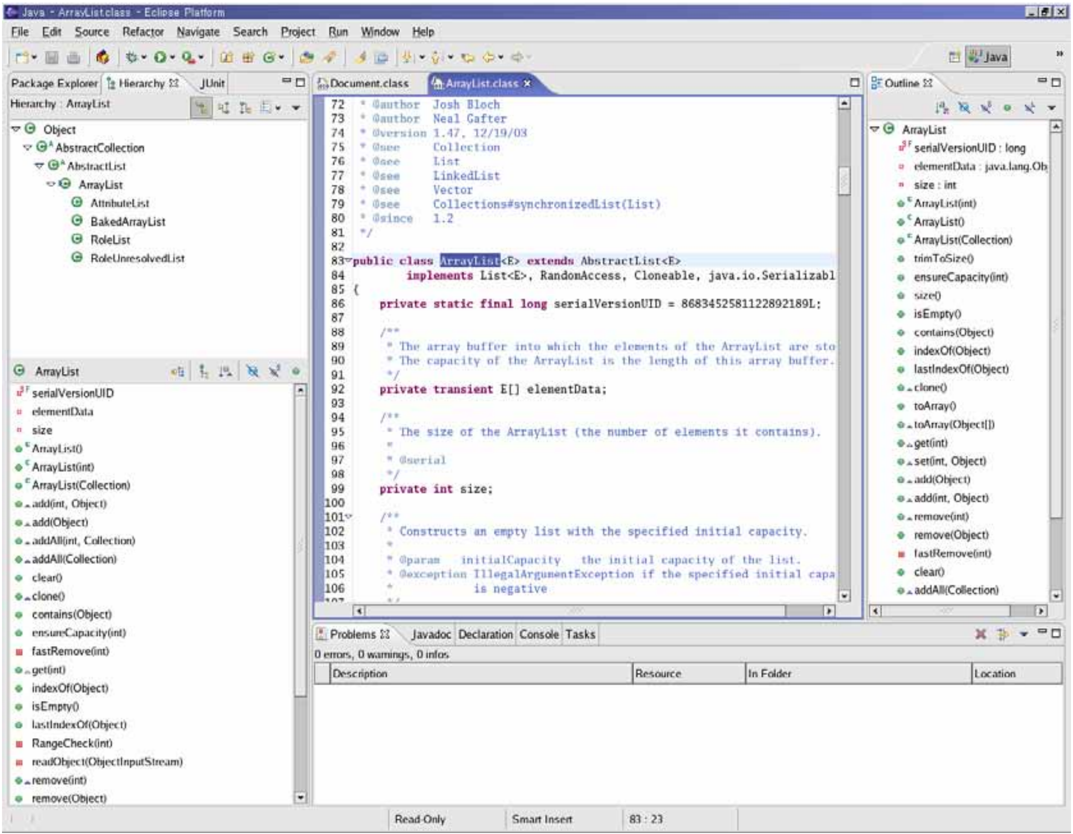


그림 6-4. [Hierarchy]보기

[Hierarchy]보기의 웃부분에는 형사이의 계승관계(형계층)가 계층구조로 현시된다. [Hierarchy]보기의 아래부분에는 [Hierarchy]보기에서 선택된 형의 성원(마당이나 메

쏘드)이 목록으로 현시된다. [Hierarchy]보기의 도구띠에서 ▼단추를 찰칵 →[layout]에서 이러한 2개의 부분화면을 나란히 배치시킬수도 있고 형계층만을 현시할수도 있다. 이미 열려진 [Hierarchy]보기에서 AbstractList를 오른쪽찰칵하여 [Focus on 'AbstractList']을 선택하면 AbstractList의 형계층이 현시된다(그림 6-5). 이와 같이 어떤 형에 초점을 주면 그 형에 부분형이 존재하는 경우에는 모든 부분형들을 현시할수 있다.

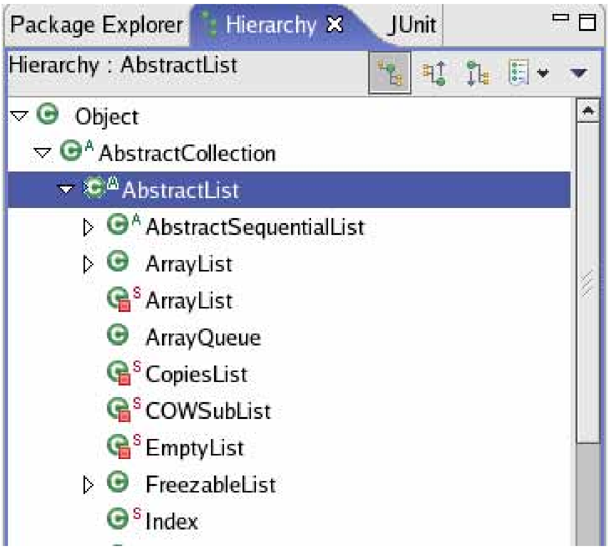


그림 6-5. AbstractList의 형계층

[Show the Supertype Hierarchy]단추를 찰칵하면 초점이 있는 형의 상위형만이 현시된다. [Show the Subtype Hierarchy]단추를 찰칵하면 초점이 주어져있는 형의 부분형만이 현시된다. [Previous Type Hierarchies]단추를 찰칵하면 현시된 형의 리력이 현시되기때문에 그로부터 선택되여 이전에 현시되였던 형계층을 다시 현시하게 할수 있다. [Hierarchy]보기에서 형을 선택하면 아래쪽의 화면에 그 형의 성원이 현시된다. [Hierarchy]보기에서 AbstractList클라스를 선택하여 현시되는 성원목록에서 add(object)메쏘드를 선택한다. 성원목록의 화면에 있는 [Lock View and Show Members in Hierarchy]단추를 찰칵하여 선택된 상태로 하면 [Hierarchy]보기에서 add(object)메쏘드가 어느 클라스에서 덧쓰기(overwrite)되였는가가 현시된다(그림 6-6). 성원목록화면의 웃부분에는 그밖에 표 6-1에 제시한 단추들이 있어 현시시킨 대상을 교체할수 있다.

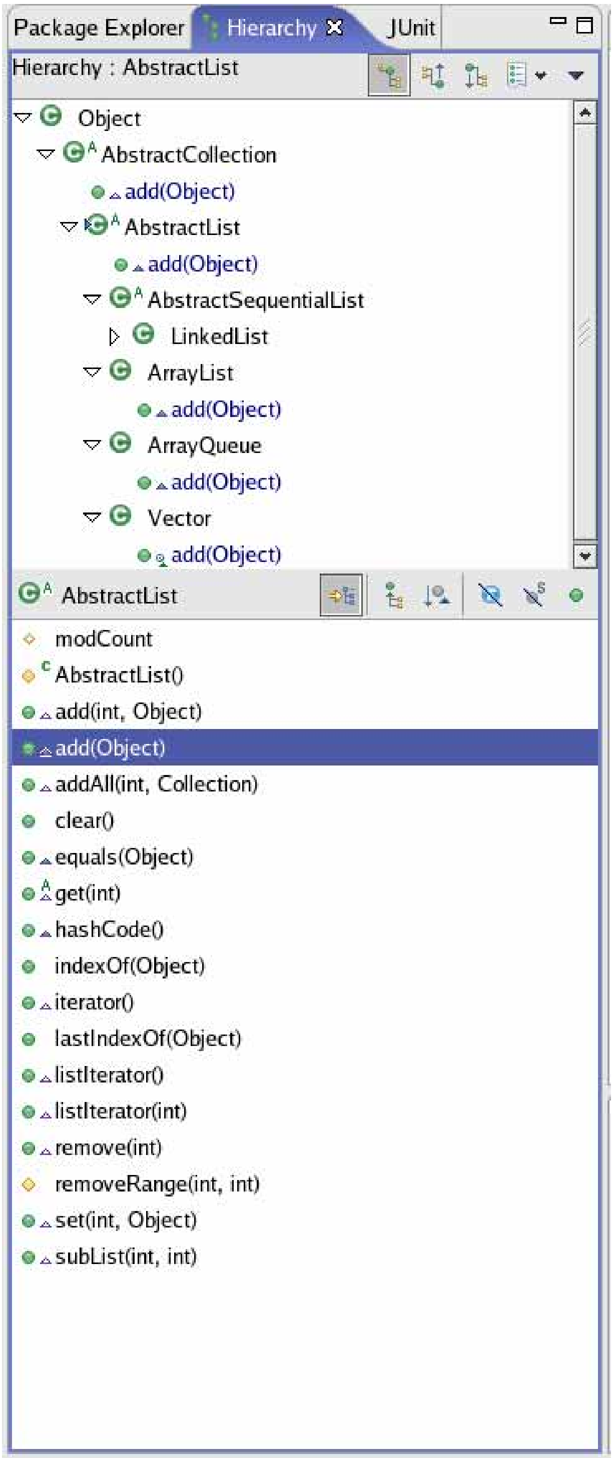


그림 6-6. [Lock View and Show Members in Hierarchy] 단추를 선택한 경우

표 6-1. 성원목록화면의 단추

단 추	단추를 눌렀을 때의 현시
Lock View and Show Members in Hierarchy	선택된 성원을 [Hierarchy]보기에 현시한다.
Show All Inherited Members	선택되여있는 클라스에 정의되여있는 성원만이 아니라 어미클라스에서 정의되여있는 성원도 현시한다.
Sort Methods by the Defining Type	메쏘드의 되돌림값의 형으로 메쏘드를 정렬하여 현시한다.
Hide Fields	마당이 현시되지 않게 한다.
Hide Static Fields and Methods Static	static인 마당이나 메쏘드를 현시하지 않게 한다.
Hide Non-Public Members	public마당이나 메쏘드만을 현시하게 한다.

지금까지 [Hierarchy]보기를 열었을 때에 형을 하나만 선택할수 있었다. 또한 패키지나 서류철에 대하여서도 형계층을 현시할수 있다. [Package Explorer]보기에서 [SampleChap6]프로젝트→[test]서류철을 선택하고 F4건을 누르면 그 서류철내의 모든 클라스에 대하여 형계층이 현시된다(그림 6-7). 서류철이나 패키지의 형계층을 현시한 경우는 선택된 서류철 또는 패키지안의 형들에는 풀색표식, 그 이외의 형에는 백색표식이 붙는다.

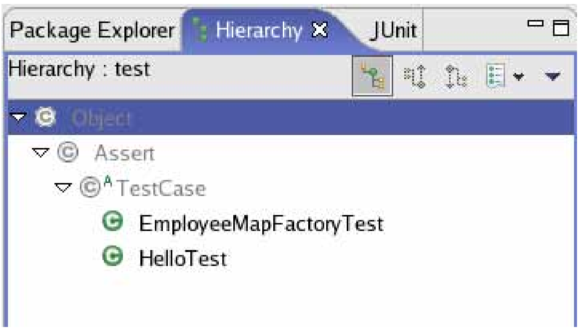


그림 6-7. 패키지전체의 형계층현시

차림표띠의 [Window]→[Open Perspective]→[Others]를 선택하여 현시된 대화칸에서 [Java Type Hierarchy]을 선택하면 [Java Type Hierarchy]투시가 열린다(그림 6-8). 또한 지름차림표의 [Open Perspective]단추로도 [Java Type Hierarchy]투시를 열수 있다. 이 투시는 [Hierarchy]보기와 편집기령역으로 단순한 구성을 이루기때문에 형계층을 찾으면서 원천코드를 참조하는데 편리하다.

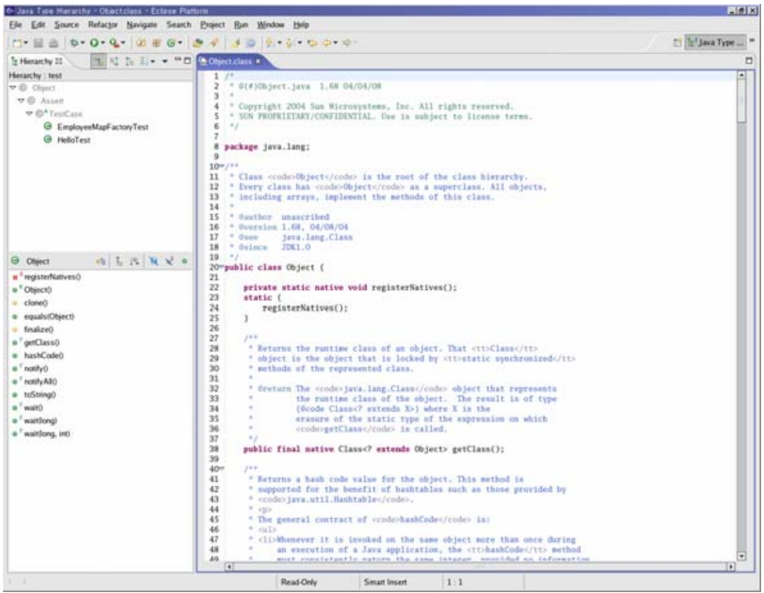


그림 6-8. [Java Type Hierarchy]투시

제3절. Java검색, 선언의 참조

차림표띠에서 [Search]→[Java]를 선택하면 그림 6-9와 같은 대화칸이 현시된다. [Java Search]태브에서 문자렬을 지정하여 클라스나 마당의 선언, 참조하려는 위치 등을 검색할수 있다. 검색대상목록을 표 6-2에, 검색대상의 제한목록을 표 6-3에, 검색범위를 표 6-4에 주었다.

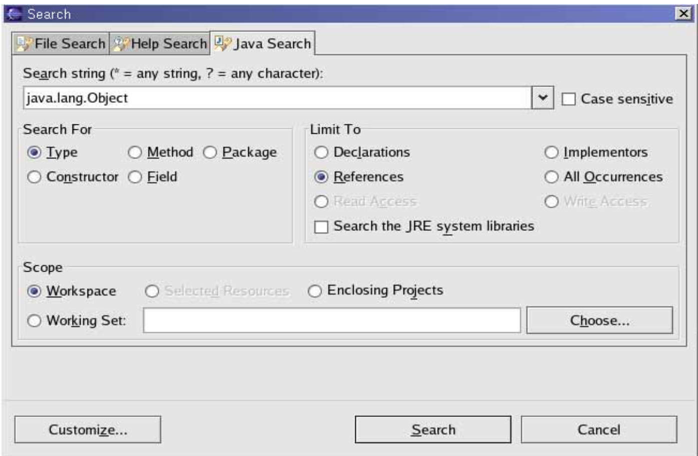


그림 6-9. [Java Search]대화칸

표 6-2. [Search For]그룹의 선택항목들

선택항목	설명
Type	[Search string]에 입력한 이름의 클라스 또는 대면부를 검색
Method	[Search string]에 입력한 이름의 메쏘드를 검색
Package	[Search string]에 입력한 이름의 패키지를 검색
Constructor	[Search string]에 입력한 이름의 구성자를 검색
Field	[Search string]에 입력한 이름의 마당을 검색

표 6-3. [Limit To]그룹의 선택항목들

선택항목	설 명	비고
Declaration	클라스, 메쏘드, 마당들이 정의되여있는 위치를 검색	
Interface	대면을 실현하고있는 클라스를 검색	[Search For]그룹에서 [Types)]을 선택하였을 때만 유효
Reference	클라스, 메쏘드, 마당 등이 사용되고있는 위치를 검색	
All Occurrennces	선언, 구현, 참조의 모든 것을 검색	
Read Access	마당을 참조하고있는 위치를 검색	[Search For]그룹에서 [Field]을 선택할 때만 유효
Write Access	마당을 변화시키고있는 위치를 검색	[Search For]그룹에서 [Field]을 선택할 때만 유효
Search JRE System Libraries	검사할 때 JRE체계서고를 검색범위에 포함	[Limit To]그룹에서 [Reference] [All Occurrennces]를 선택하였을 때만 설정가능

표 6-4. [Scope]그룹의 선택항목들

선택항목	설 명
Workspace	현재의 작업공간전체를 검색
Selected Resources	현재 선택된 자원만을 검색
Enclosing Project	현재 선택된 자원이 포함되여있는 프로젝트전체를 검색
Work Set	선택된 작업조에서 정의된 범위를 검색(작업조에 대하여서는 6장의 제4절 작업조를 참고)

[Java]투시의 [Package Explorer]보기에서 [SampleChap6]프로젝트를 선택한다. 앞에서 본 방법대로 차림표에서 [Java Search]태브를 열고 SampleChap6프로젝트에서 TestCase클라스가 참조되고있는 장소를 검색해보기로 한다. 그러자면 [Java Search]태브의 [Search String]마당에 《 TestCase 》를 입력하고 [Search For]그룹에서 《Types》을 선택한다.

[Limit To]그룹에서 [References]를 선택하고 [Scope]그룹에서 [Selected Resoures]을 선택한다. [Search]단추를 찰칵하면 검색이 시작되여 [Java]투시의 오른쪽아래의 [Search]보기에 결과가 현시된다(그림 6-10). [Search]보기의 [Next Match]단추나 [Previous Match]단추를 찰칵하면 검색에서 대조된 부분의 원천코드가 편집기에 현시된다. 이때 대조된 문자렬이 반전현시되는데 그 행이 노란색의 화살표로 제시된다. [Search]보기의 검색결과는 프로젝트, Java패키지, 파일, 형들로 그룹화되여 있다. 각각 [Group by Project],[Group by Package], [Group by File], [Group by Type]의 단추를 찰칵하는것으로서 그룹들사이의 절환을 실현할수 있다.

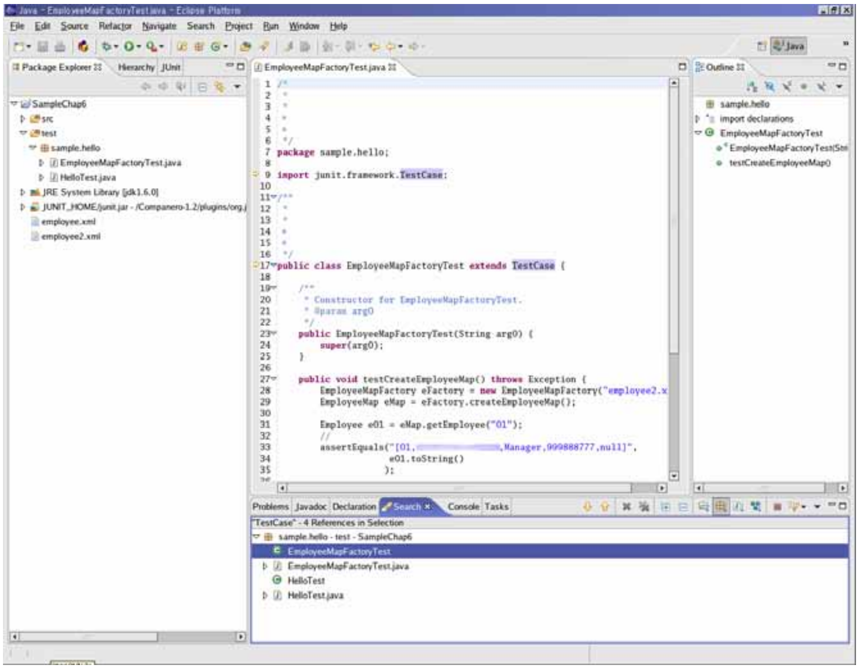


그림 6-10. 검색결과를 현시하는 [Search]보기

검색은 지름차림표에서도 진행할수 있다. 여기서는 DisplayObject대면을 실현하고있는 클라스를 검색하여 본다. [Package Explorer]보기에서 [SampleChap6]프로젝트→[src]서류철→[sample.hello]패키지→[DisplayObiect.Java]→[DisplayObiect]를 선택하고 마우스오른쪽단추를 찰칵한다. 이때 현시되는 지름차림표에서 [References]→[Workspace]를 선택하면 작업장에서 DisplayObject를 실현하고있는 클라스가 검색

되여 검색결과가 검색보기에 현시된다. [Outline]보기에서 DisplayObject를 오른쪽찰칵하여 [References]→[Workspace]을 선택하여도 꼭같이 검색할수 있다.

[Java Search]를 사용하면 마당이나 형이 선언되여있는 위치를 검색하는것이 가능하지만 선언을 검색할 때에는 보다 편리한 방법이 있다. [Package Explorer]보기에서 [SampleChap6]프로젝트→[src]서류철→[sample.hello]패키지→[Employee.Java]를 두번찰칵하여 원천코드를 현시한다. Implements구문의 [DisplayObject]를 선택하고 마우스오른쪽단추를 찰칵하고 [Open Declaration]를 선택하면 DisplayObject가 정의되여있는 원천코드가 Java편집기에 현시된다. 메쏘드나 마당에 대하여서도 이와 같이 하여 원천코드에서 정의되여있는 장소를 검색하고 그 부분을 편집기에 현시할수 있다.

또한 선언부분의 원천코드를 참조하기만 한다면 Java편집기에서 형(또는 메쏘드나 마당)우에 마우스지시자를 맞추었을 때 나타나는 튀여나오기창에 원천코드를 현시할수 있다. 또한 Java편집기에서 선언을 참조하는 형이나 메쏘드에 마우스유표를 이동시키면 [Declaration]보기에 대상의 원천코드가 현시된다(그림 6-11). 선언에서 정의된 Javadoc는 [Javadoc]보기에 현시된다(그림 6-12). 원천코드가 첨부되여있다면 건립경로에 포함되여있는 서고의 원천코드나 Javadoc도 참조할수 있다. 이 기능들을 사용하면 참조만을 하려는 경우에는 Java편집기를 열지 않아도 된다. 이렇게 Java편집기에는 편집하려는 코드만을 현시하게 함으로써 필요없이 코드들을 변경시키는것을 막을수 있게 된다.

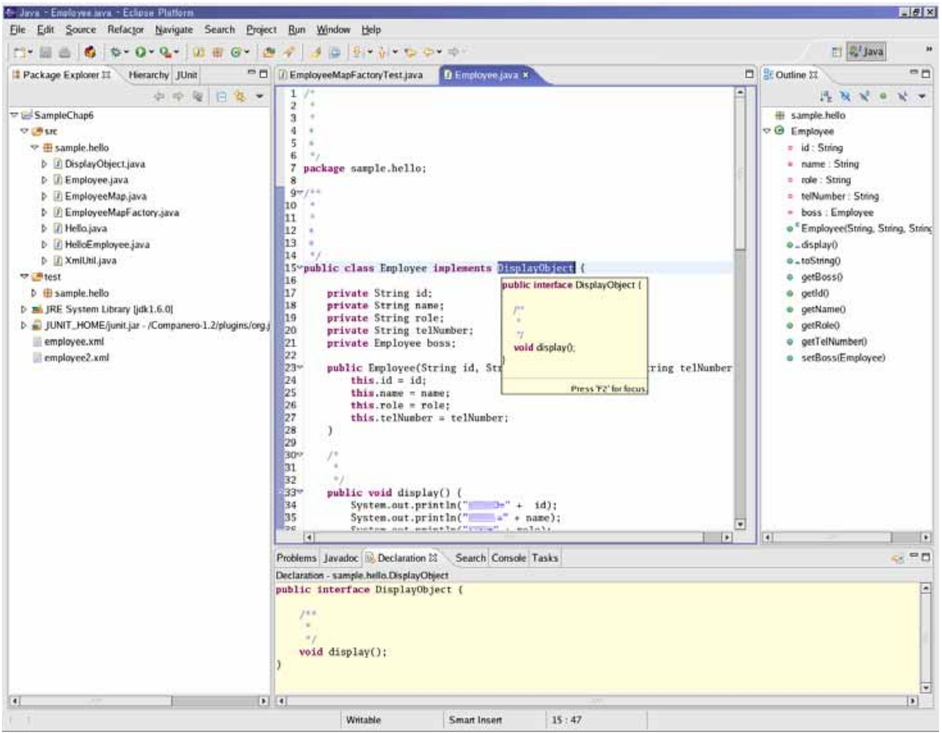


그림 6-11. [Declaration]보기에 대상의 원천코드가 현시.

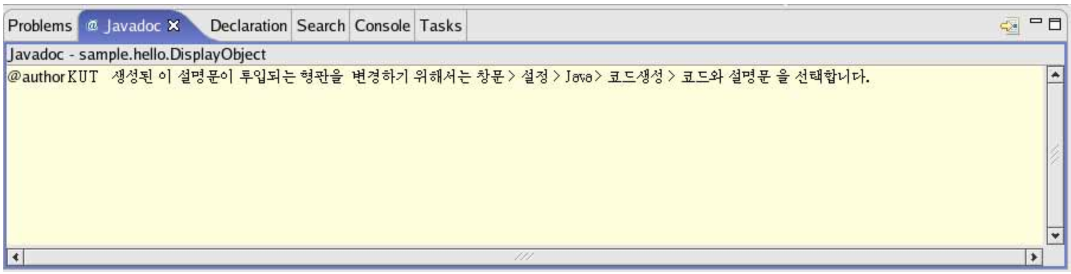


그림 6-12. Javadoc의 현시

제4절. 작업조

작업조(working set)를 사용하면 보기에서 현시하려는 요소를 제한할수 있다. [Java]투시를 열고 [Package Explorer]보기의 ▼단추를 찰칵하여 [Select Working Set]을 선택하면 그림 6-13과 같은 대화칸이 열린다.

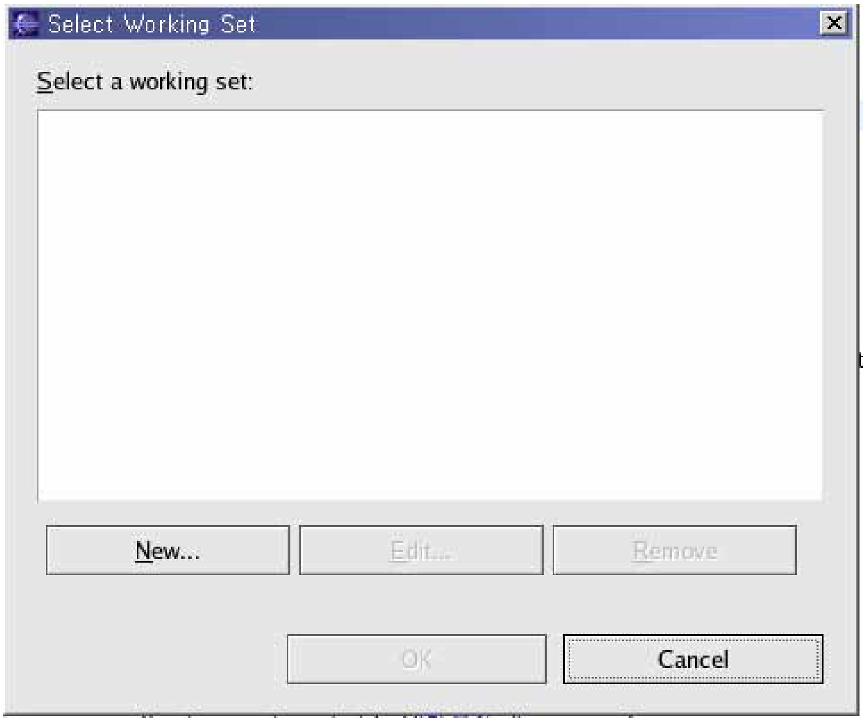


그림 6-13. 작업조의 선택

[New]단추를 찰칵하여 새로운 작업조를 설정하는 대화칸이 열린다 (그림 6-14).

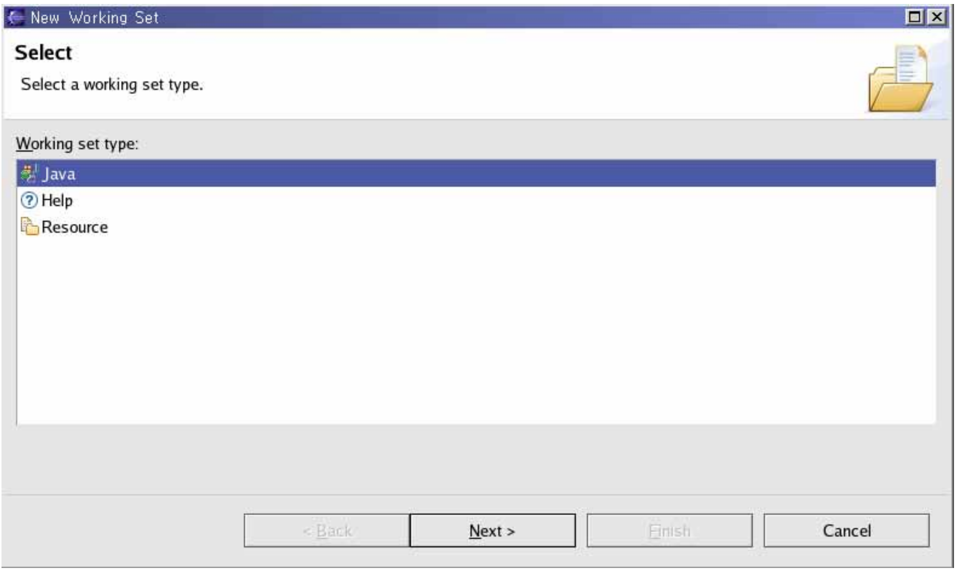


그림 6-14. 작업조설정대화칸 1

여기서 [Working set type]의 [Java]를 선택한 다음 [Next]단추를 찰칵한다. [Java Working Set]대화칸의 [Working set name]에 《SampleSrc》을 입력하고 [Working set content]에서 [SampleChap6]프로젝트를 열어 src서류철만을 선택한다(그림 6-15).

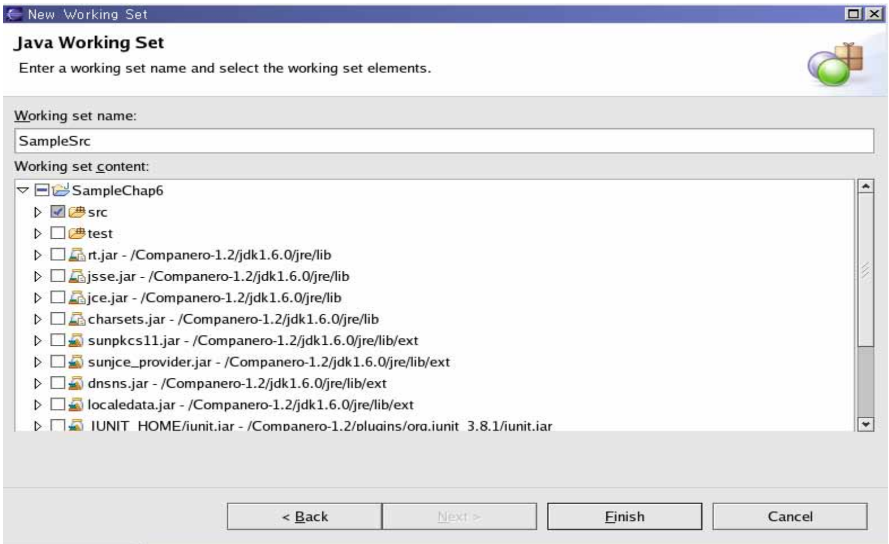


그림 6-15. 작업조설정대화칸 2

[Finish]단추를 찰칵하면 작업조의 선택화면에 돌아가 이미 작성된 SampleSrc가 목록에 현시된다. 작성된 작업조를 선택하여 [OK]단추를 찰칵하면 [Package Explorer]보기에는 이미 [Working set content]에서 선택된 내용만이 현시되게 된다(그림 6-16). 원래대로 되살리기 위해서는 [Package Explorer]보기의 ▼단추를 찰칵하여 [Deselect Working set]를 선택한다.

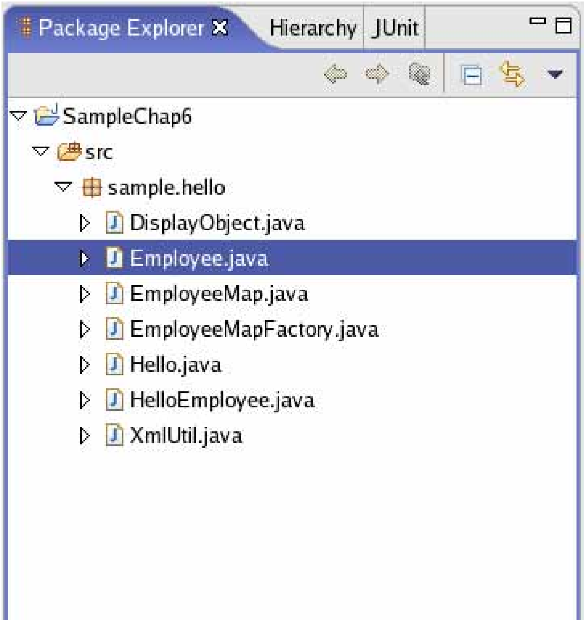


그림 6-16. 작업조작성 후의 패키지열람기

작업조의 작성은 [Package Explorer]보기뿐만아니라 [Navigator]보기와 [Task]보기의 현시대상을 조절하려고할 때에도 리용한다. [Java Search]에서 [Scope]의 설정을 [Workspace]으로 하였을 때와 [Working set]으로 설정하였을 때 java.util.Map의 참조를 검색하여 결과의 차이를 확인해본다(그림 6-17).

작업조는 모든 보기에서 다 설정할수 있으며 어떤 보기에서 작업조를 설정한것이 다른 보기에 영향을 주지 않는다. 또한 작업조를 설정한 후에 추가되는 요소가 기존요소의 하위요소이면 자동적으로 추가되지만 그렇지 않는 경우는 작업조를 편집하여 추가하지 않는 한 자동적으로 추가되지 않는다.

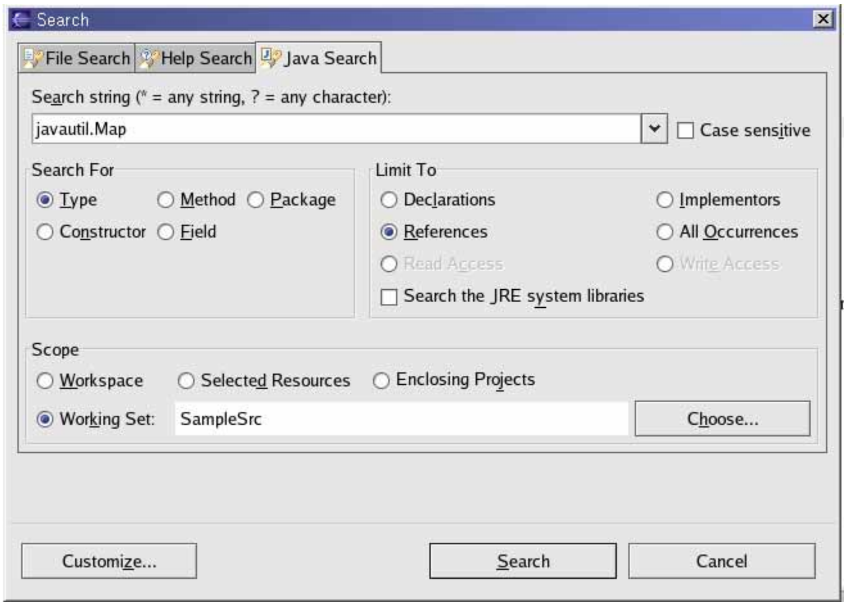


그림 6-17. [Java Search]에서 작업조를 사용

제5절. 문자렬의 외부화

특정한 나라나 지역에서만 사용하는 응용프로그람이면 차림표나 통보문을 그 나라나 지역에서 사용하고있는 언어로 현시해도 별문제가 없겠지만 여러 나라들이나 지역에서 사용되는 응용프로그람이면 매개 언어에 대응할 필요가 있다. 또한 언어만이 아니라 날자의 표기에서도 《1 April 2003》과 같이 《일, 월, 년》의 순서인 경우나 《2003년 4월 1일》과 같이 《년, 월, 일》순서로 된 경우 등 나라마다 차이가 있다. 응용프로그람을 여러 나라의 언어, 화폐와 수값 등에 대응시키는것을 응용프로그람의 외부화라고 한다.

외부화시켜야 할 응용프로그람에서는 나라나 지역에 맞추어 화면에 현시되는 통보문이나 차림표, 단추의 명칭 등을 절환시킬 필요가 있다. 또한 원천코드에 이 문자렬들이 직접 넣어져있으면 언어마다 다른 원천코드가 필요하게 된다. 이것을 극복하기 위해 나라나 지역별로 속성파일을 준비하여놓고 응용프로그람은 속성파일에서 문자렬을 취하도록 하는것이 일반적인 수법으로 되고있다. Eclipse에서 《문자렬의 외부화》를 사용하여 원천코드를 수정하는것은 조수를 리용하면 간단히 진행할수 있다

Java투시의 패키지열람기에서 [SampleChap6]프로젝트를 오른쪽찰칵하여 [Source]→[Externalize Strings]을 선택하면 그림 6-18과 같은 문자렬이 들어있는 원천코드의 목록이 현시된다. 이 목록에서 《Hello.Java》를 선택하고 [Externalize]를 찰칵하면 그림 6-19와 같이 문자렬의 외부화조수대화칸이 현시된다. [String to externalize]에서 행을 선택하면 [Context]령역에 원천코드의 할당부분이 반전현시되기때문에 어떤 문자렬이 외부화되고있는가를 확인할수 있다.

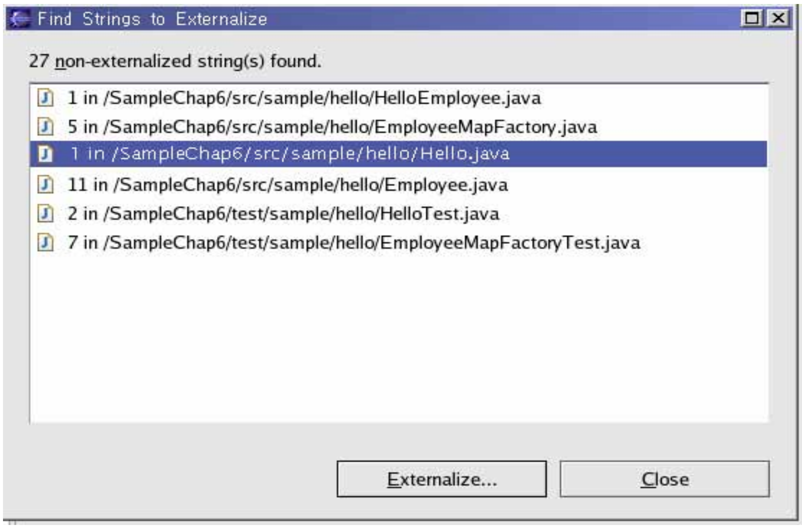


그림 6-18. 외부화하려는 문자렬의 선택

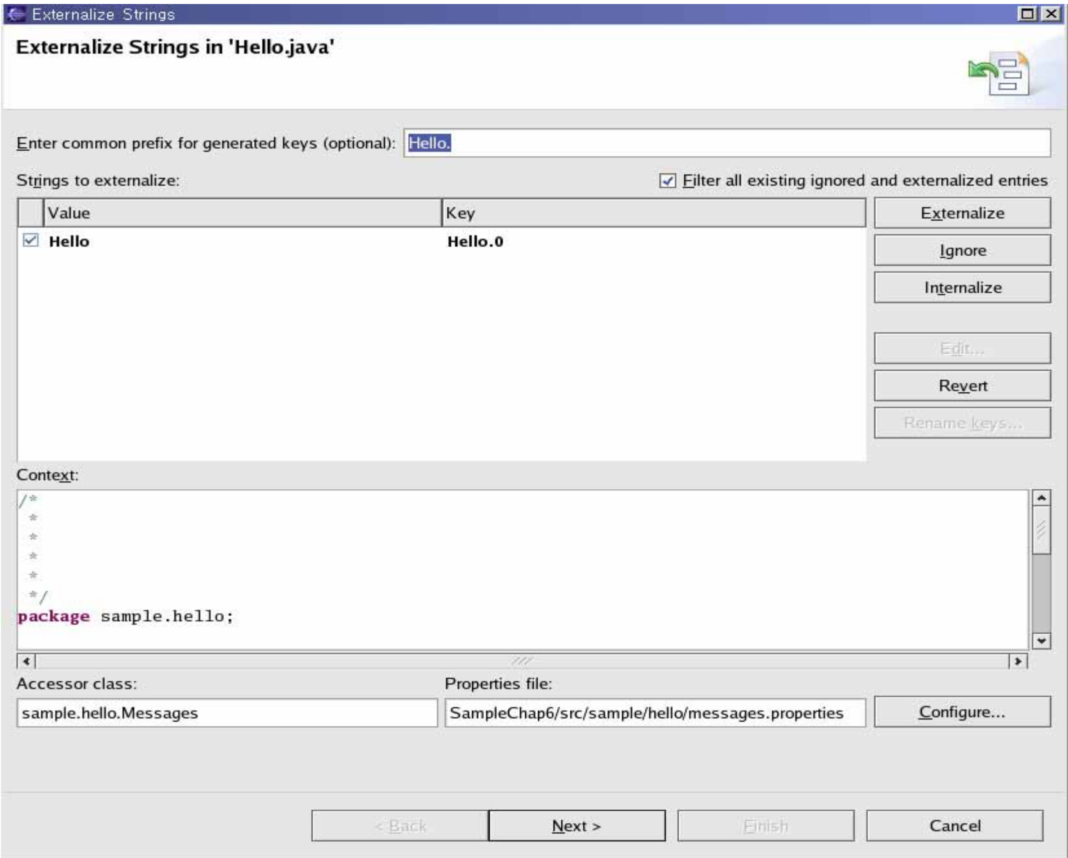


그림 6-19. 문자렬의 외부화조수대화칸

대화칸의 바로 아래에 호출기클라스와 속성파일의 이름이 현시된다. 기정현시되는 클라스나 속성파일을 변경하려는 경우에는 [configure]를 찰칵한다. 여기서는 기정클라스와 속성파일을 그대로 리용함으로써 [Next]단추를 찰칵한다. 경고할 문제가 제기되면 대화칸(그림 6-20）에 통보문이 현시되며 특별히 문제가 제기되지 않으면 이 대화칸은 현시되지 않는다. [Next]단추를 찰칵한다.

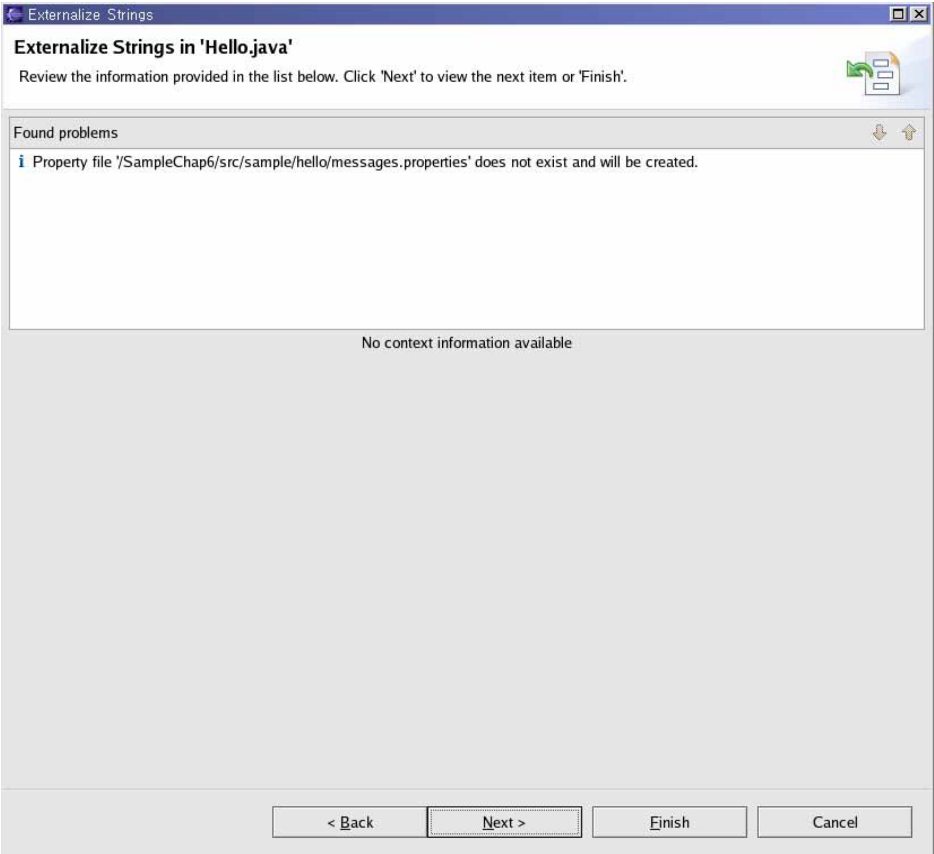


그림 6-20. 경고통보문의 현시

다음 대화칸에서는 원천코드에서의 변경된 부분을 사전에 확인할수 있다（그림 6-21）. 확인한 다음 [Finish]단추를 찰칵한다. 그러면 [Externalize Strings]대화칸이 현시되는데 [Close]단추를 찰칵하면 된다.

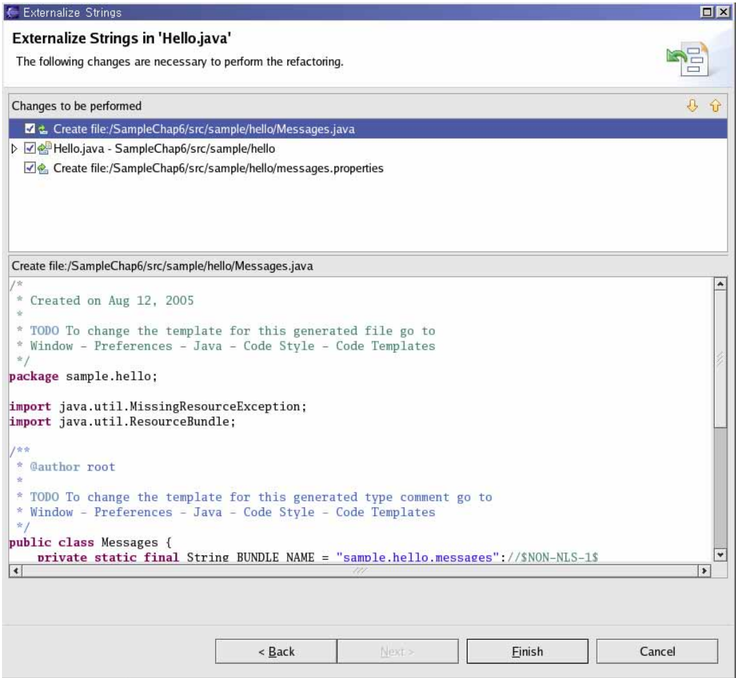


그림 6-21. 변경이 진행된 원천코드의 확인

이상의 조작에 의해 Hello.Java에 변경이 진행되며 Messages.Java(목록 6-1)와 Messages.Properties(목록 6-2)의 2개 파일이 작성된다. 이 변경으로 하여 문제가 발생하지 않는가를 확인해보기로 한다. [Package Explorer]보기에서 [SampleChap6]프로젝트→[test]서류철→[sample.hello]패키지→[HelloTest.java]를 선택하고 마우스 오른쪽단추를 찰칵한다. 현시되는 지름차림표에서 [Run]→[Junit Test]을 선택하여 실행시켜 본다.

목록 6-1. Messages.java

```
/*
*작성날자  2005년: XX월 XX일
*
*TODO이 생성된 파일의 형판을 변경하기 위해 뛰여넘기:
*창문-설정-Java-코드격식-코드형판
*/
package sample.hello;
import java.util.MissingResourceException;
import java.util.ResourceBundle;
/*
*@auther
```

```
 *
 *TODO이 생성된 t형해설문의 형판을 변경하기 위해 뛰여넘기:
 *창문-설정-Java-코드격식-코드형판
 */
 public class Messages{
 private static final String BUNDLE_NAME=
                 "sample.hello.messages";// $NON-NLS-1$
 private static final ResourceBundle Resource_BUNDLE=
             ResourceBundle.getBundle(BUNDLE_NAME);
     private Messages(){
     }
     public static String getString(String key){
         //TODO자동생성된 메쏘드
         Try{
         return Resource_BUNDLE.getString(key);
         } catch(MissingResourceException e) {
             return '!' +key+ '!';
         }
     }
 }
```

목록 6-2. messages.properties

```
 Hello.0=Hello
```

문자렬의 외부화를 취소하기 위해서는 차림표띠에서 [Refactor]→[Undo]를 선택한다. 실제로 실행하여 원래대로 되는가를 확인하여 본다.

우의 례에서는 외부화하는 문자렬에 영문자밖에 포함되지 않았다. 조선어나 한자, 일본어같이 ASCII문자외의 문자가 속성파일에 서술되여있는 경우에는 《/udddd》의 형식으로 표기되는 Unicode문자를 사용해야 한다.(J2SE의 SDK에 부속되여있는 native2ascii라는 도구를 사용하면 민족코드의 파일을 Unicode문자로 변환할수 있다. 또한 -reverse선택항목을 붙혀서 그 반대(Unicode전위문자→민족코드)의 변환도 진행한다.)

Eclipse의 [Externalize Strings]를 사용하여 조선어문자렬을 외부화하면 자동적으로 Unicode전위문자에로의 변환도 진행된다. [Package Explorer]보기에서 [SampleChap6]프로젝트→[src]서류철→[sample.hello]패키지→[Employee.java]를 오른쪽찰칵하여 [Source]→[Externalize Strings]를 선택하면 문자렬의 외부화대화칸이 현시된다(그림 6-22). 열쇠 [Employee.0]의 값이 [종업원ID=]인가를 확인해본다. 다음 [Next]단추를 찰칵하여 확인화면까지 나간다. 확인화면의 [Changes to be performed]에서 [Create file:/SampleChap6/src/sample/hello/messages.properties]을 선택하여 messages.properties의 내용을 현시한다(그림 6-23).

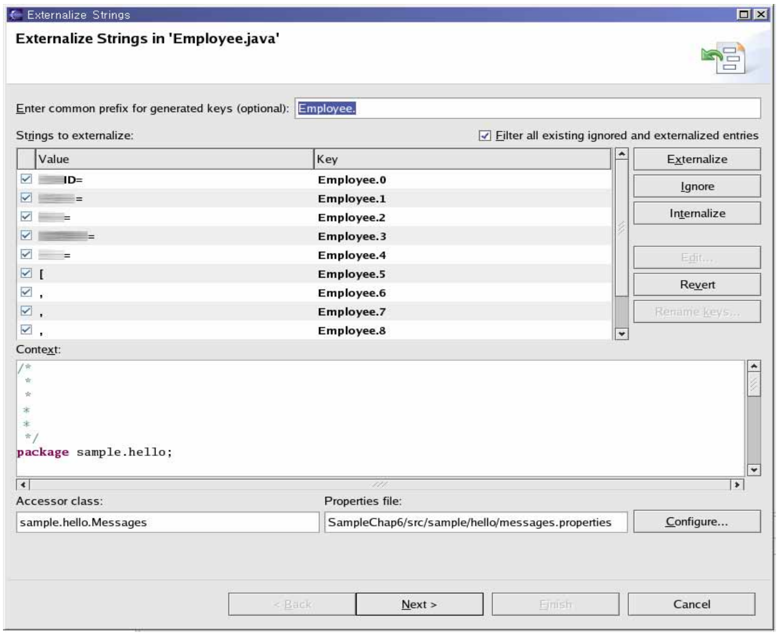


그림 6-22. 조선어문자렬의 외부화

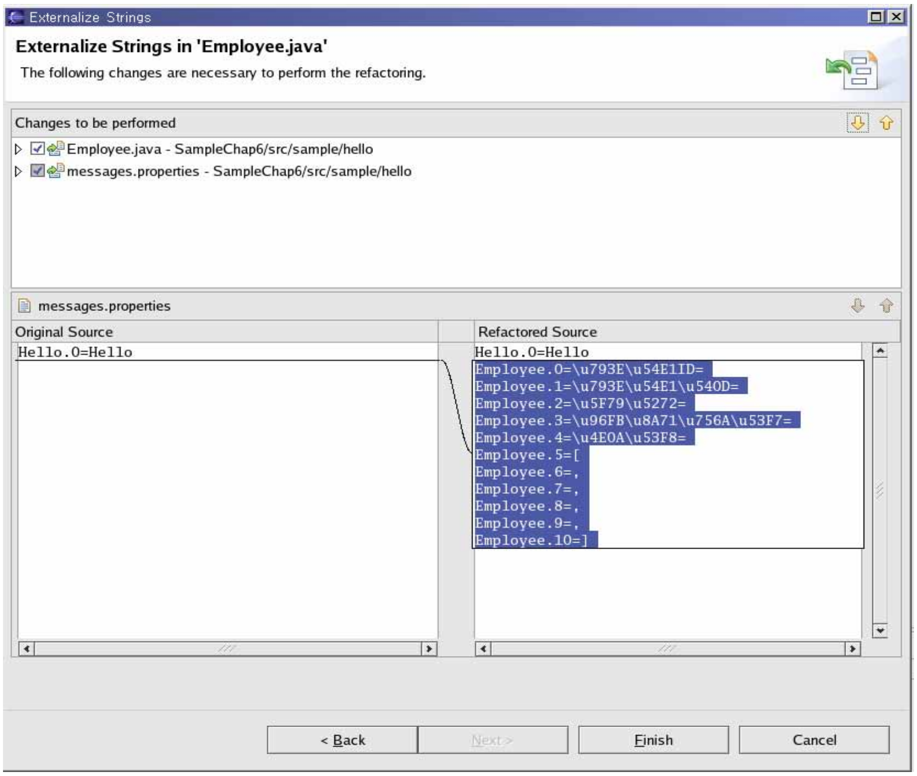


그림 6-23. 조선어문자렬을 포함하는 속성파일

Employee.3은 [전화번호=]라는 문자렬이였으나 [전화번호]가 《/u96FB/u8A71/u756A/u53F7》으로 변화되는것을 확인할수 있다. 이렇게 하여도 프로그람의 실행에서는 지장이 없지만 속성파일의 읽기를 쉽게 하기 위하여 열쇠에 의미가 있는 이름을 달아주는것이 좋다. 그림 6-22의 대화칸에서 열쇠를 선택한 다음 [Edit]단추를 찰칵하여 현시되는 대화칸에서 열쇠의 문자렬이나 대응하는 값을 직접 편집할수 있다.

제6절. 오려둠책(scrapbook)

오려둠책을 사용하면 완전한 클라스의 원천코드를 작성하지 않고도 코드의 어떤 토막만을 실행시킬수 있다.

오려둠책를 사용하기 위해서는 먼저 오려둠책을 작성하여야 한다. 차림표띠에서 [File]→[New]→[Others]를 선택하여 현시된 대화칸에서 [Java]→[Java Run/Debug]→[Scrapbook Page]를 선택한 다음 [Next]단추를 찰칵한다. 그러면 그림 6-24와 같은 대화칸이 현시된다.

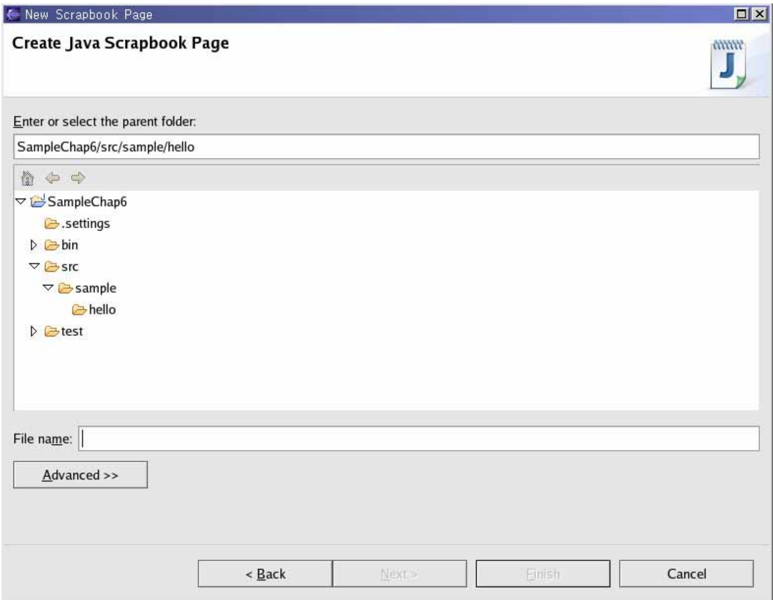


그림 6-24. 오려둠책페지의 새로 작성

SampleChap6프로젝트를 선택한 다음 [File name]마당에 《myscrap》라고 입력하고 [Finish]단추를 찰칵한다. 그러면《myscrap.jpage》라는 빈 편집기가 열린다. 이 편집기에 목록 6-3의 코드토막을 입력한다.

목록 6-3.

```
sample.hello.Hello hello=new sample.hello.Hello();
System.out.println(hello.getgreeting( "***" ));
```

입력된 코드토막전체를 선택하고 오른쪽찰칵하여 [Run]을 선택한다. 코드토막이 실행되여 console보기에 [Hello ***]라고 현시된다. 객체나 값을 되돌려주는 메쏘드의 되돌이값을 안다면 console보기에 출력하는 코드를 쓰지 않고도 지름차림표에서 [Checkout]나 [Display]를 선택하여 비교해볼수 있다. 이와 같이 오려둠책은 여러 행의 원천코드를 시험적으로 실행하여보려고 하는 경우에 사용할수 있다.

제7절. 표준코드작성

한 사람이 작성하는 프로그람이라면 코드작성격식은 별로 문제될것이 없다. 자신의 의도에 맞는 격식으로 코드작성을 하면 그만이다. 원천코드가 어떻든 구문적으로 차이가 없는 한 아무런 문의도 없이 콤파일이 진행된다. 그러나 쏘프트웨어개발프로젝트의 성원으로서 프로그람을 작성하는 경우에는 프로그람작성자에 따라 코드작성격식이 차이나는데 이것은 문제로 된다.

팀으로 개발하는 경우 다른 사람이 작성한 코드를 읽을 필요가 제기된다. 이때 코드작성형태가 각이하면 코드를 읽기 힘들고 다시 보는데 시간이 더 많이 소모되는 등 여러가지 결함들을 피할수 없게 된다. 프로젝트에서 코드작성표준을 정해놓으면 모두가 표준에 따라 코드작성을 함으로써 팀전체의 개발능률을 높일수 있다. 개발수법의 하나인 XP(eXtreme Programming)수법에서는 코드작성표준이 실현되여있다. 여기서는 코드작성표준화를 지원하는 Eclipse의 기능에 대하여 소개한다.

6.7.1. 코드서식기(code formatter)

[Preference]대화칸에서 Eclipse가 자동생성하는 코드나 Javadoc, 서식화(format)를 진행할 때에 적용되는 코드작성격식을 지정할수 있다. Eclipse 3.0에서는 이전 보다 코드서식기가 더 강력해졌으며 세밀하게 설정할수 있게 되였다.

코드서식기의 설정은 [Preference]대화칸에서 진행한다. 차림표띠에서 [Window]→[Preference]을 선택한다. 현시되는 대화칸의 왼쪽에서 [Java]→ [Code style]→[Code Formatter]를 선택한다(그림 6-25).

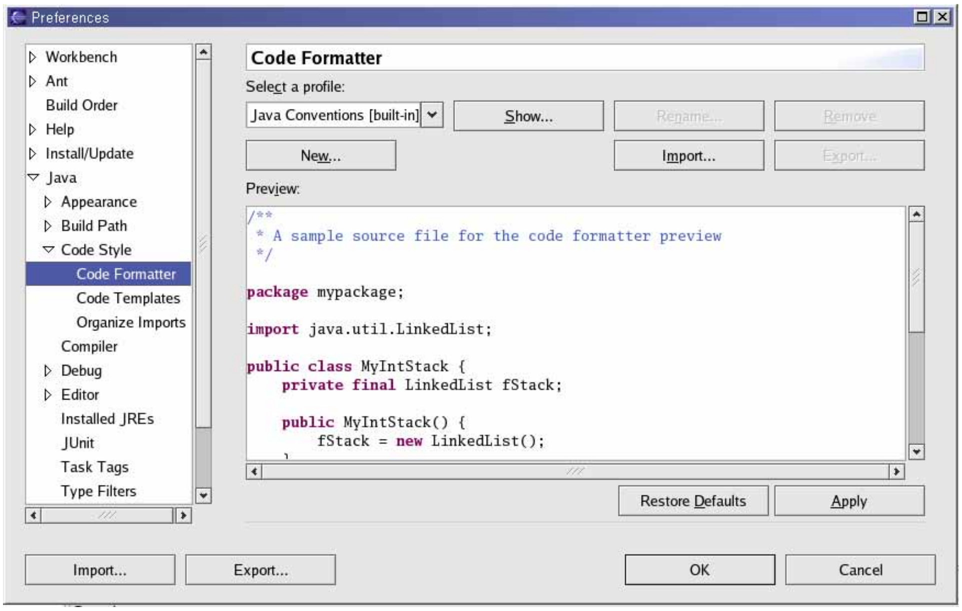


그림 6-25. 코드서식기의 설정대화칸

기정으로는 프로파일로서 [Java Conventions [built-in]]이 선택되여있다. 이 밖에 [Eclipse 2.1[built in]]을 선택할수 있다. 선택한 프로파일에 의해 어떻게 서식화되는가는 [Preview]에서 확인할수 있다. 이밖의 서식화를 진행하려는 경우에는 새롭게 프로파일을 정의할수 있다. 그러자면 대화칸에서 [New]단추를 찰칵하여 그림 6-26의 대화칸을 연다.



그림 6-26. 새로운 프로파일의 작성

[Profile name] 에 《Sample Profile》을 입력하고 [Initialize settings with the following profile]에서 《Java Conventions [built-in]》을 선택한 다음 [OK]단추를 찰칵한다. 이상의 설정으로 기정의 Java규약을 기초로 프로젝트의 코드작성표준에 맞도록 서식기를 설정하였다(그림 6-27).

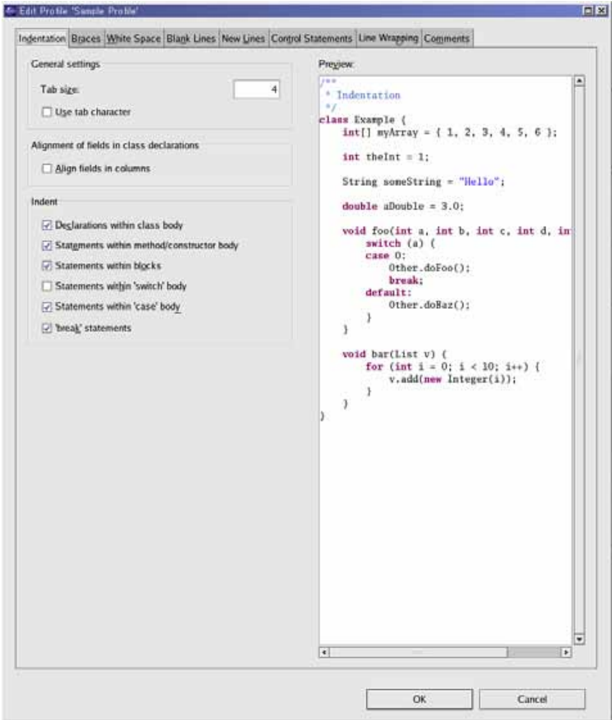


그림 6-27. 코드서식기에서 프로파일의 편집

그림 6-27에서 설정을 변경하려는 태브를 선택하고 화면의 현시에 따라 변경한다. 거의 대부분의 항목을 검사칸의 선택이나 내리펼침목록의 선택으로 설정할수 있다. 어떻게 서식화되는가는 매 태브에 있는 [Preview]령역에서 확인할수 있다.

표 6-5에 설정가능한 항목들가운데서 일부를 소개하였다.

표 6-5. 코드서식기 프로파일의 설정항목

태브이름	설정내용
Indentation	들여쓰기에서 사용하는 공백의 수, 들여쓰기에 태브를 사용하는가, 들여쓰는 위치의 설정을 진행
Braces	블로크시작을 나타내는 괄호의 설정(선언과 같은 행, 선언의 다음 행 등)
White Space	공백의 설정(if와 괄호사이에 공백을 넣는 등)
Blank Lines	공백행의 설정(패키지선언의 다음, 메쏘드선언하기 전에 공백행을 넣겠는가 하는 위치설정과 그 행의 개수설정)
New lines	빈 블로크나 배렬초기화에서의 행바꾸기설정
Control Statements	조종명령문(if else,try-catch 등)의 행바꾸기설정
Line Wrapping	한 행의 최대문자수설정이나 최대문자수를 넘어날 때 행바꾸기의 설정
Comments	설명문부분의 서식설정

여기서는 클라스나 메쏘드의 정의나 if-else문, try-catch-finally문 등의 블로크를 표시하는 중괄호(《{》, 《}》)의 앞뒤에서 행바꾸기를 설정하였다. 그림 6-27의 대화칸에서 [Braces]태브를 선택한다. 여기서 [Array initializer(배렬초기화기)]를 제외한 나머지는 [Next line]으로 변경시킨다(그림 6-28).

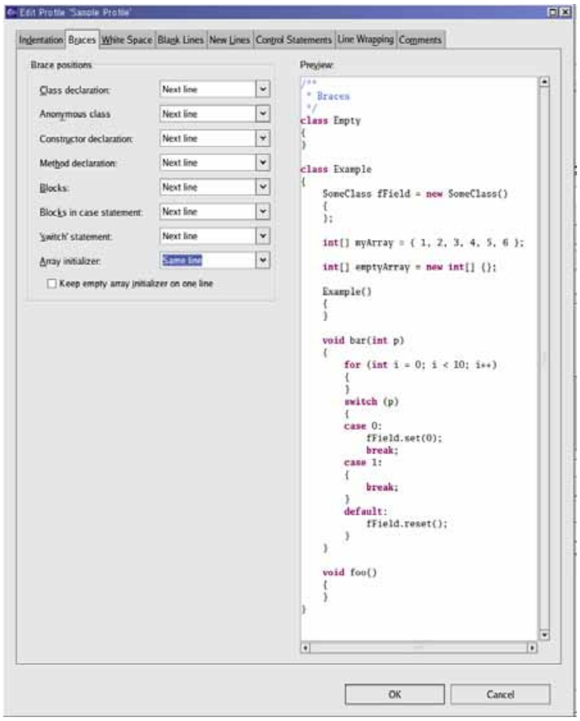


그림 6-28. [Braces]태브의 설정

다음으로 [Control statements]태브를 선택하고 [General]그룹에서 [Insert new line before 'while' in a 'do' statement('do'명령문안의 'while'명령문의 앞에 행바꾸기를 삽입)]을 제외하고 나머지는 모두 선택한다(그림 6-29). [OK]단추를 찰칵하여 설정대화칸을 닫고 프로파일에 [Sample Profile]이 선택되여있는가를 확인한 다음 [Preference]대화칸도 닫는다.

서식기를 실행하기 위해서는 차림표에서 [Source]→[Format] 또는 [Package Explorer]보기를 오른쪽찰칵→[Source]→[Format]을 선택한다. [Package Explorer]보기에서 프로젝트나 원천서류철을 선택한 경우에는 그의 자원에 포함되는 원천코드전체가 서식화된다. 한개의 원천코드를 서식화한 경우에는 차림표띠의 [Edit]→[Undo]를 반복하는것으로서 원래대로 되돌려 보낼수 있지만 여러개의 원천코드에 대하여 서식화를 진행하였으면 서식화전의 상태로 되돌아가지 않는다. 서식화를 실행하기 전에 경고통보문이 있기때문에 그 시점에서 중지할수 있다는데 주의를 돌려야 한다.

[Package Explorer]보기에서 [SampleChap6]프로젝트 → [src] → [EmployeeMapFactory.java]를 선택하여 우와 같은 순서로 서식화를 실행하여본다. 이미 설정된 프로파일에 따라 서식화되는가를 확인해보아야 한다.

팀 전체가 같은 코드작성표준에 따라 작성하기 위해서는 모두가 같은 서식화프로파일을 사용하면 된다. Eclipse에서 설정한 서식화프로파일은 XML격식으로 반출, 반입할수 있다. 반출이나 반입을 진행하려면 [Preference]대화칸의 [Code Formatter](그림 6-25)에 있는 [export]단추나 [import]단추를 사용한다. 이 단추를 리용하여 팀에서 같은 서식프로파일을 사용하며 CVS 등의 보관고에서 선택하기 전에 반드시 Eclipse의 코드서식기를 일치시키는것이 좋다.

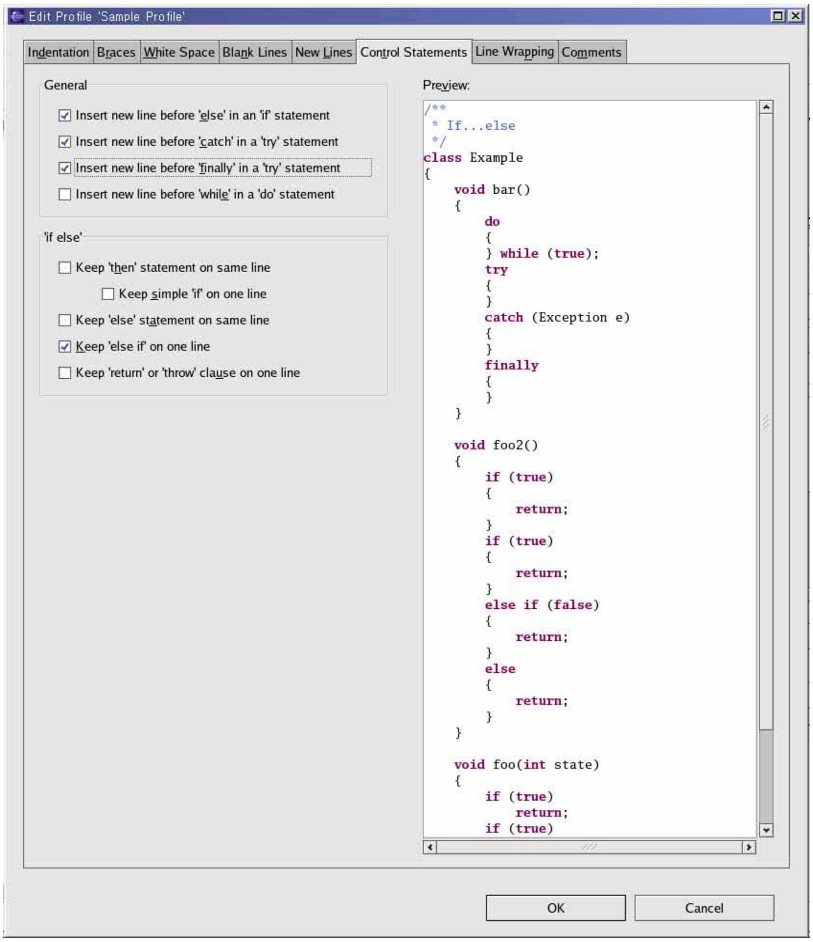


그림 6-29. [Control statements]태브의 설정

6.7.2. Eclipse Checkstyle Plug-in

코드작성표준에는 들여쓰기나 행바꾸기 등의 규칙외에 메쏘드이름이나 마당이름 등의 이름짓기규칙이 포함된다. Eclipse의 코드격식의 설정에서는 이름짓기규칙을 위한 처리를 강제로 할수 없다. 그러나 [Preference]대화칸의 [Java]→[Code style]에서 변수의 앞붙이나 뒤붙이의 이름규칙을 설정할수 있다. 이름짓기규칙의 검사에 리용할수 있는 도구로서 Checkstyle(http://checkstyle.sourceforge.net/)이 있다. Checkstyle을 사용하면 이름짓기규칙만이 아니라 들여쓰기나 행바꾸기 등의 코드작성표준건반에 관한 검사를 할수 있다. Checkstyle을 Eclipse에서 실행할수 있게 하는 추가쏘프트웨어가 Eclipse Checkstyle Plug-in (http://eclipse-cs.sourceforge.net/)이다. Checkstyle Plug-in은 검사대상을 프로젝트단위로 설정한다.

여기서는 SampleChap6프로젝트에서 Checkstyle을 사용해보기로 한다. 실현방법은 다음과 같다.

① Java투시의 [Package Explorer]보기에서 [SampleChap6]을 오른쪽찰칵하여

나타나는 차림표에서 [Properties]을 선택한다. 그러면 프로젝트의 속성설정대화칸이 열린다.

② 대화칸에서 [Checkstyle]을 선택한다.

③ 대화칸의 오른쪽에 있는 [Add]단추를 선택한다(그림 6-30).

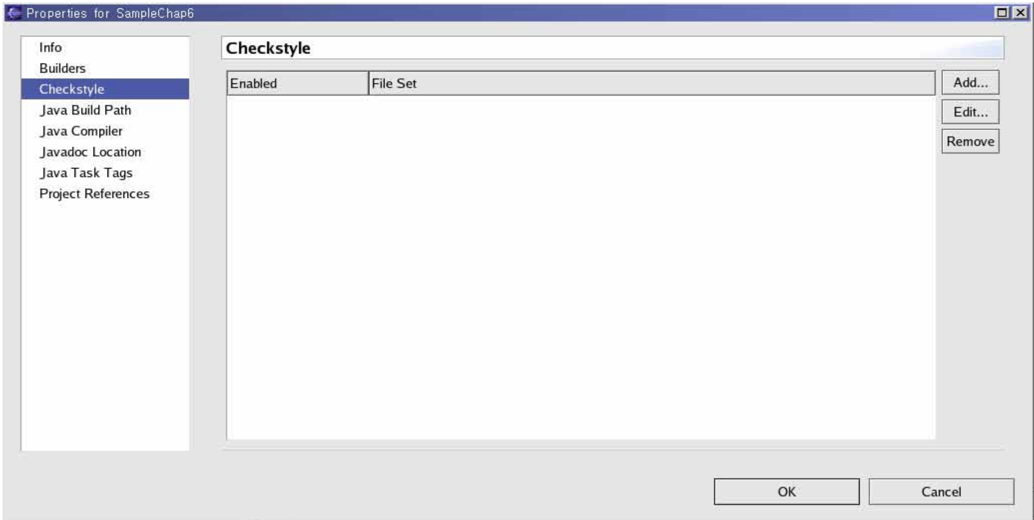


그림 6-30. SampleChap6의 속성설정화면

④ [Checkstyle File Set Editor]대화칸(그림 6-31)에서 [File Set Name]에 《All Java resource》이라고 입력하고 [Check Configuration]에서 《Sun Checks》를 선택한다.(기정으로는 [Sun Checks]밖에 선택하지 못한다. [Sun Checks]를 선택하면 Sun의 코드작성규약에 기초하여 검사를 진행한다.)

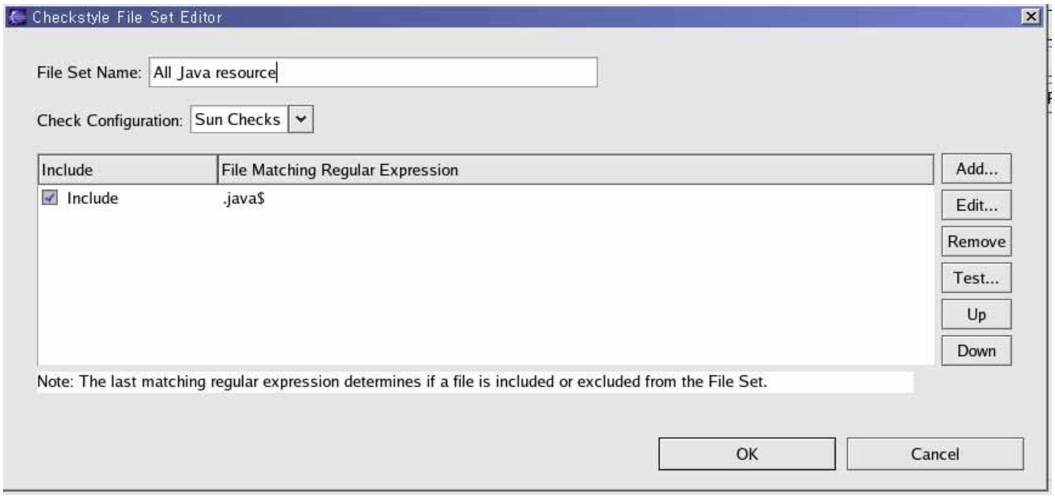


그림 6-31. [Checkstyle File Set Editor]대화칸

⑤ 대화칸의 아래에는 검사대상으로 하는 파일표현형식을 정규표현으로 설정할수 있다. [Add]단추를 찰칵하면 정규표현을 추가할수 있다. [Test]단추를 찰칵하면 어떤 파일이 검사대상으로 되는가가 목록으로 현시되므로 의도한대로 파일이 포함되여있는가를 확인할수 있다. 여기서는 기정(파일이름의 제일 끝이 java인 파일)대로 설정하고 [OK]단추를 찰칵하여 대화칸을 닫는다.

⑥ 프로젝트의 속성설정대화칸에서 이자 등록한 파일묶음이 능동으로 되여 있는가를 확인하고 [OK]단추를 찰칵한다.

우와 같은 설정에 의해 Checkstyle이 자동적으로 실행되여 코드작성표준에 맞지 않는 부분이 Java편집기나 [Package Explorer]보기 등에 현시된다(그림 6-32). 실제로 여러개의 원천코드를 현시하여놓고 어떤 경고가 나오는가를 확인하여 본다. 행마지막문자가 공백으로 되고 Javadoc태그가 없는 등 세밀한 구문까지 검사한다는것을 알수 있다.

[Problems]보기의 [Filter]단추를 찰칵하면 현시되는 대화칸에서 [Show itmes of type]의 [Checkstyle Marker]에 선택표식을 한 다음 [OK]단추를 찰칵하면 [Problems]보기에 Checkstyle의 경고가 현시된다.

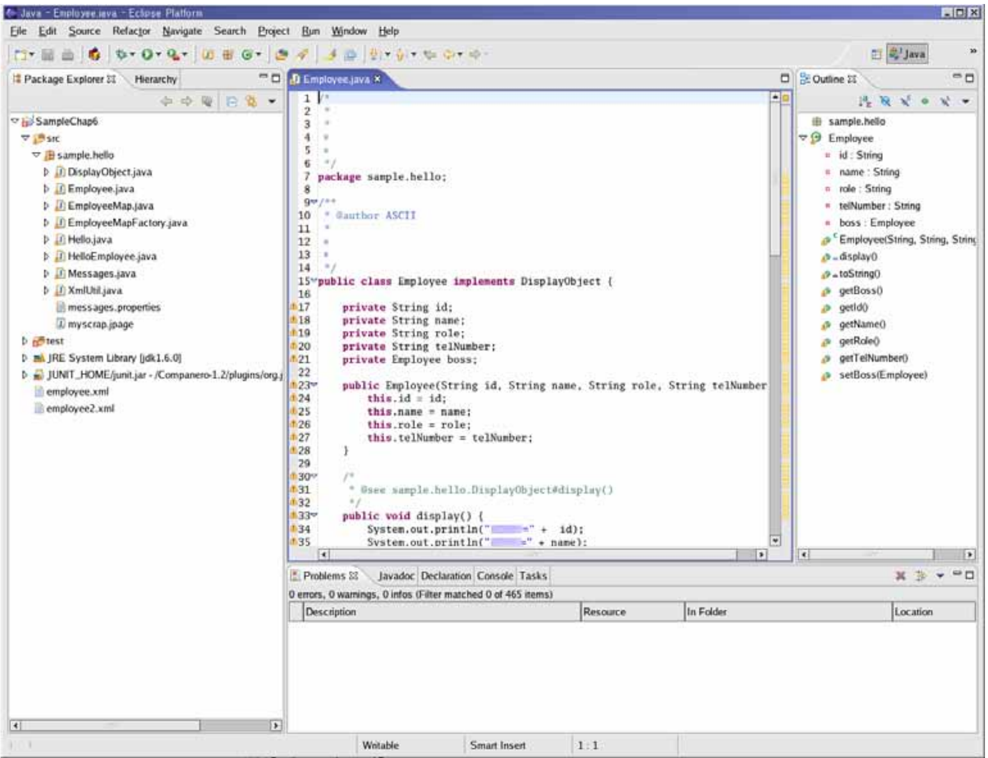


그림 6-32. Checkstyle의 경고

Sun의 코드작성규약이 아니라 프로젝트의 코드작성표준에 따르는 검사를 하고싶은 경우는 Checkstyle의 설정을 진행해야 한다. 차림표띠의 [Window]→[Preference]에서 [Preference]대화칸을 열고 Checkstyle을 선택한다(그림 6-33). 기정으로는 이미 설정한 [Sun Checks]가 등록되여있다.

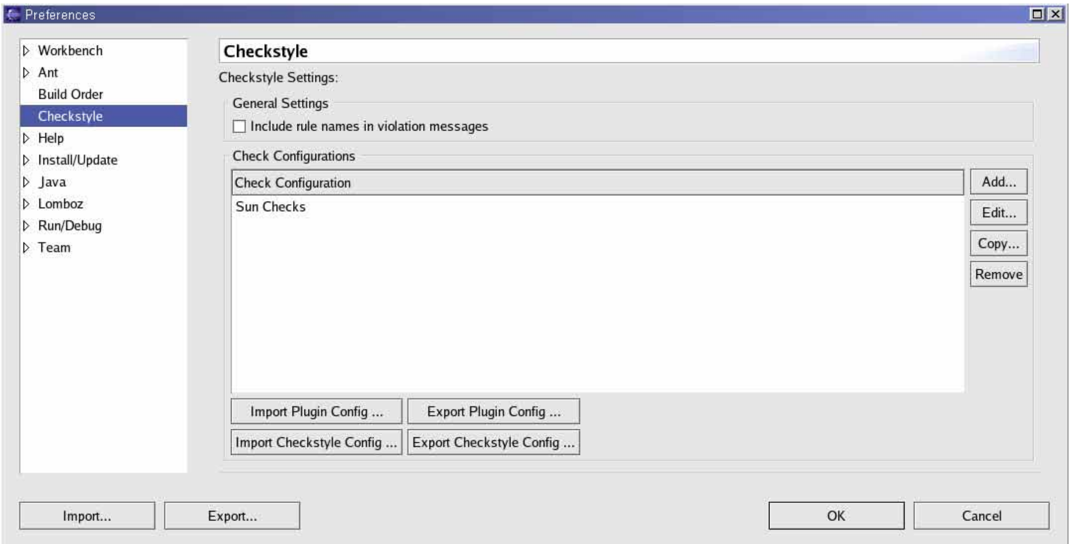


그림 6-33. Checkstyle의 설정

팀에서 Checkstyle을 리용하고있고 작성한 Checkstyle의 설정파일이 있을 때 이 대화칸을 사용하여 반입하면 Eclipse에서 리용할수 있다. 이 경우 [Import Checkstyle Config]단추를 찰칵하여 설정파일을 읽어들인다. [Import Plugin Config]단추는 [Export Plugin Config]단추를 사용하여 반출된 파일을 반입하는 경우에 사용한다. 이것은 Checkstyle끼워넣기의 설정이 원래의 Checkstyle설정을 확장하고있는것이기때문에 팀전체가 Eclipse Checkstyle Plug-in을 리용하는 경우는 이것들을 사용하여 설정을 공유하는것이 좋다.

또한 [Export Plugin Config]단추를 사용하면 Eclipse의 외부에서 Checkstyle을 실행하는 때에 리용하는 보통의 설정파일을 반출할수 있다. [Add]단추를 찰칵하여 [Checkstyle Check Configuration Editor]대화칸을 열고 새롭게 Checkstyle의 검사규칙을 설정할수 있다(그림 6-34).

[Checkstyle Check Configuration Editor]에서는 내리펼침목록으로부터 설정하는 항목을 선택할수 있는 등의 입력지원기능이 있고 원래의 Checkstyle보다도 간단히 검사규칙을 편집할수 있게 되였다. Sun의 코드작성규약을 토대로 하여 프로젝트용설정을 작성한것이면 Checkstyle의 설정대화칸에서 [Sun Checks]를 선택하여 [Copy]단

추를 찰칵한다. 이렇게 하면 [Sun Checks]의 검사규칙이 복사된 상태에서 [Checkstyle Check Configuration Editor]가 열리기때문에 필요한 곳만 수정할수 있어 품을 줄일수 있다. 기정의 [Sun Checks]에서 규약의 위반은 경고를 현시하게 되여 있지만 이것을 오유로 현시하게 할수도 있다. 자세한 내용은 Checkstyle의 설명서(도움말) 등을 참고한다.

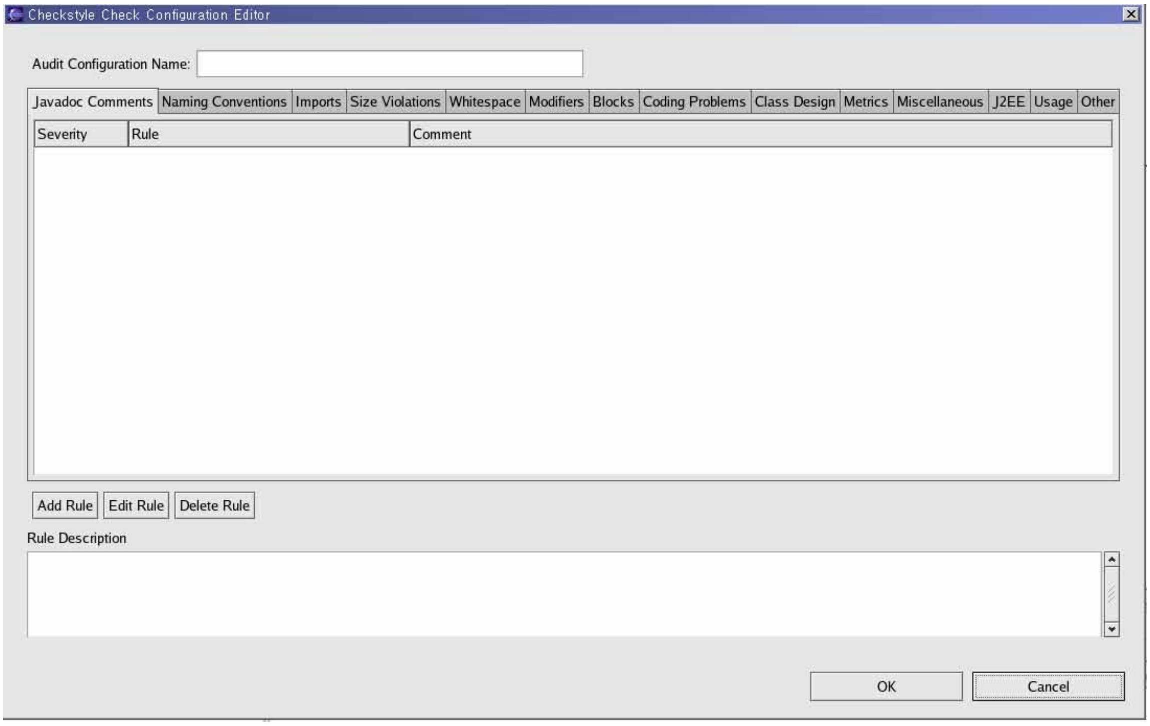


그림 6-34. 검사방식 및 검사구성편집기

제7장. 개 편

제1절. 개편의 정의

개편(Refactoring)이란 쏘프트웨어의 외적거동을 그대로 보존한 상태에서 내부구조를 개선해나가는것을 말한다.

구체적으로는 코드를 살펴보다가 길이가 긴 메쏘드, 크기가 큰 클라스의 분할, 리해하기 힘든 클라스, 메쏘드이름의 변경, 공통기능을 상위클라스에로 보내기와 같은 조작들을 진행하여 코드를 개선하는것을 말한다.

보통 체계개발에서는 일단 완성되여 시험을 통과한 코드는 될수록 다치지 않고 수정할 필요가 생겨도 변경하지 않는 방법을 택하는것이 대부분이다. 이러한 환경에서 개편을 적극적으로 하는것은 좀 어렵다. 그러나 최근에 주목되고있는 [XP(eXtreme Programming)]에서 개편의 실시가 적극적으로 제창되고있고 반복형의 개발수법이 일반적으로 쓰이게 됨에 따라 개편도 자연히 개발작업의 일환으로 되여가고있다.

제2절. 개편에 필요한것

7.2.1. 재귀시험묶음

개편은 바로 코드의 수정이다. 코드를 수정할 때 프로그람의 거동에는 변화를 주지 말아야 한다. 이를 위하여 《이 시험에서 합격하면 이 코드는 완전히 정확하다.》고 확신할수 있는 정확한 시험묶음(test suite)이 필요하다. Eclipse에는 이미 소개된것처럼 단체시험도구인 JUnit가 들어있고 시험경우(test case)/시험묶음(test suite)의 작성지원기능이 갖추어져있을뿐아니라 Ant기능과 조합하면 구축할 때마다 자동적으로 시험을 실행할수 있다.

7.2.2. 개편도구

클라스나 마당, 메쏘드의 이름 변경, 메쏘드의 파라메터 변경, 다른 클라스에로의 메쏘드의 이동과 같은 개편을 진행하기 위하여 변경할 요소가 들어있는 부분을 빠짐없이 찾아내여 적절하게 변경시키야 하며 또한 많은 클라스를 다른 패키지로 이동하려는 경우 package문을 바꾸어쓸 뿐만아니라 원천파일을 파일체계에서 이동시키야 한다. 이와 같이 단순하면서도 시간이 걸리는 작업은 콤퓨터에서 진행하는것이 제일 빠르고 정확하다. Eclipse에는 Java문법을 정확하게 리해하는 성능이 높은 검색엔진과 사용빈도가 높은 개편조작을 반자동화하는 개편제공기능이 갖추어져있다.

회귀시험의 자동화를 지원하는 JUnit와 정확도가 높은 개편을 진행할수 있는 도구를 갖추고있는 Eclipse는 개편을 적극적으로 진행하기 위한 리상적인 개발환경이라고 말할수 있다.

제3절. 개편의 실례

Eclipse의 개편제공기능을 충분하게 활용한 개편조작에 대하여 실례를 들어 설명하겠다. 이를 위해서 SampleChapter7프로젝트를 생성하고 설명을 진행하도록 한다.

7.3.1. StringEncoder클라스

SampleChapter7프로젝트는 간단한 문자렬부호화기(string encoder)의 역할을 수행하는 Java프로젝트이다. 이 부호화기는 문자렬을 받아서 그것을 지정한 방법으로 변환하는 서고로서 요구사항은 다음과 같다.

- Native2Ascii, ROT13(자모순의 매 문자를 13문자 변경시키는 간단한 암호화방식)의 두 방식에서의 부호화에 대응해야 한다.
- 새로운 부호화방식의 추가에 간단하게 대처할수 있어야 한다.

그때 기존부호화기의 원천코드에 손댈 필요가 없어야 한다.

이 요구사항을 만족시키는 작업이 이미 완성되고 부호화기의 동작을 검증하기 위한 시험클라스도 준비되여있다(표 7-1, 그림 7-1).

표 7-1. 관련클라스일람

클 라 스	설 명
StringEncoder	부호화기의 API정의(가상메소드 “encode”)와 보조클라스의 생성기능을 가진 추상클라스
StringEncoder. Encoder	부호화방식을 나타내는 렬거형(typesafe enum) NATIVE2ASCII와 ROT13이 정의되여있다.
Native2Ascii Encoder	StringEncoder보조클라스. native2ascii지령과 같은 방법으로 문자렬을 Unicode조종문자로 변환하는 부호화기
ROT13Encoder	StringEncoder보조클라스. ROT13방식으로 문자렬을 암호화하는 부호화기
StringEncoderTest	StringEncoder의 실례생성기능을 시험한다.
Native2Ascii EncoderTest	Native2AsciiEncoder의 거동을 정의한다.
ROT13EncoderTest	ROT13Encoder의 거동을 정의한다.
EncoderMain	부호화기의 리용실례. StringEncoder를 사용하여 문자렬을 부호화한다.

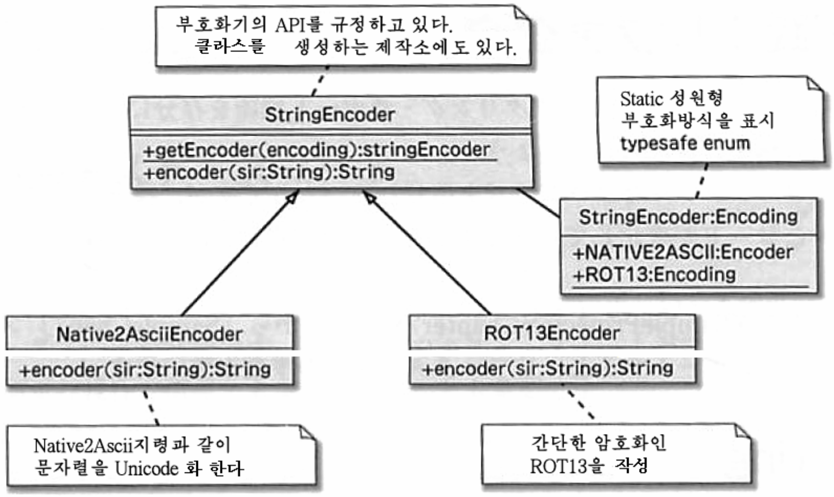


그림 7-1. 개편전의 클라스도

다음으로 부호화기의 리용방법에 대하여 보기로 한다.

```
//먼저 부호화기의 대면을 생성
StringEncoder encoder
   =StringEncoder.getEncoder(StringEncoder.Encoding.NATIVE2ASCII);
// encode메쏘드에 부호화대상문자렬을 넘기면
//부호화된 문자렬이 출력된다.
String result=encoder.encoder.encode(“통일조국one!”);
System.out.println(result);
```

출력결과는 다음과 같다.

```
/ud1b5/uc77c/uc870/uad6done!
```

7.3.2. 문제점의 도출

얼핏 보기에는 이 부호화기에 별로 문제가 없다고 생각된다. 이 부호화기는 물론 정확히 동작하며 요구사항을 만족시킨다. 또한 설계의 일반성을 만족시키고 있다고 볼수 있다. 메쏘드(getEncoder)작성으로 실현클라스(implementation class)의 이름을 모르고서도 부호화기의 대면을 생성할수 있으며 새로운 부호화기방식을 제공할 필요가 있을 때에도 실현클라스의 추가요구에 대응할수 있다. 여기에 몇가지 불합리한 점이 있는데 그 하나는 StringEncoder클라스이다. 이 클라스에는 부호화기가 실현할수 있는 API의 정의(encode메쏘드)와 부호화기의 실현클라스를 적절하게 대면화하는 제작소(factory)의 기능을 실현하는 getEncoder메쏘드가 들어있다. 이 정의로 하여 StringEncoder의 실현클라스는 본래의 부호화기로서의 거동외에 제작소로서의 역할도 계승한다. 즉

StringEncoder encoder
=Native2AsciiEncoder.getEncoder(StringEncoder.Encoding.ROT13);
다음으로 부호화기의 원천코드를 본다(그림 7-2).

<table>
<tr><td>Public String encode(String str){
Try{</td><td></td><td>Public String encode(String str){
Try{</td></tr>
<tr><td>//1.읽기준비
StringReader in=
new StringReader(str);
StringWriter out=
new StringWriter();</td><td>동일함</td><td>//1.읽기준비
StringReader in=
new StringReader(str);
StringWriter out=
new StringWriter();</td></tr>
<tr><td>//2.부호화기처리
for(int c=0;(c=in.read())!=-1;){
if (31<c && c < 127){
out.write(c);
}else{
String hex=
integer.toHexString(c);
//0을 4자리로 한다.
While (hex.length()<4){
hex= “0” +hex);
}
out.write(“\\u” +hex);
}
}</td><td>여기가 차이난다</td><td>//2. 부호화기처리
for(int c=0;(c=in.read())!=-1;){
if((‘a’ <=c && c<= ‘m’) ||(‘A’ <=c && c <= ‘M’)){c+=13;
}else if ((‘n’ <=c && c<= ‘z’) ||(‘N’ <=c &&c <= ‘Z’)){c-=13;
}
out.write(c);
}</td></tr>
<tr><td>//3.후처리
try{
in.close();
} finally {
out.close();
}

return out.getBuffer().toString();
}catch(IOException ice){
return str;
}</td><td>여기도 같다</td><td>//3.후처리
try{
in.close();
} finally {
out.close();
}

return out.getBuffer().toString();
}catch(IOException ice){
return str;
}</td></tr>
</table>

그림 7-2. 부호화기의 원천코드를 대비

2개의 부호화기원천코드는 부호화알고리듬을 서술한 부분외에는 꼭 같다. 아마 이런 클라스의 작성자는 제일 처음 어느 한 클라스를 쓰고 복사(copy)나 붙이기(paste)로 다른 하나를 만들것이다.

여기에 다음과 같은 문제점이 있다.

- StringEncoder가 API의 정의, 실현클라스의 실례화라는 두가지 서로 다른 역할을 한다.
- 부호화기의 실현클라스에 중복되는 코드가 많다.

이것을 다음과 같이 개편하기로 한다.

- 부호화기의 골격인 AbstractStringEncoder를 작성하고 매 부호화기의 공통부분을 정의
- 실현클라스의 실례화를 담당하기 위한 StringEncoderFactory를 작성
- 부호화기가 실현할수 있는 API를 정의하는 대면 StringEncoder를 작성

개편한 다음의 클라스도는 그림 7-3과 같다.

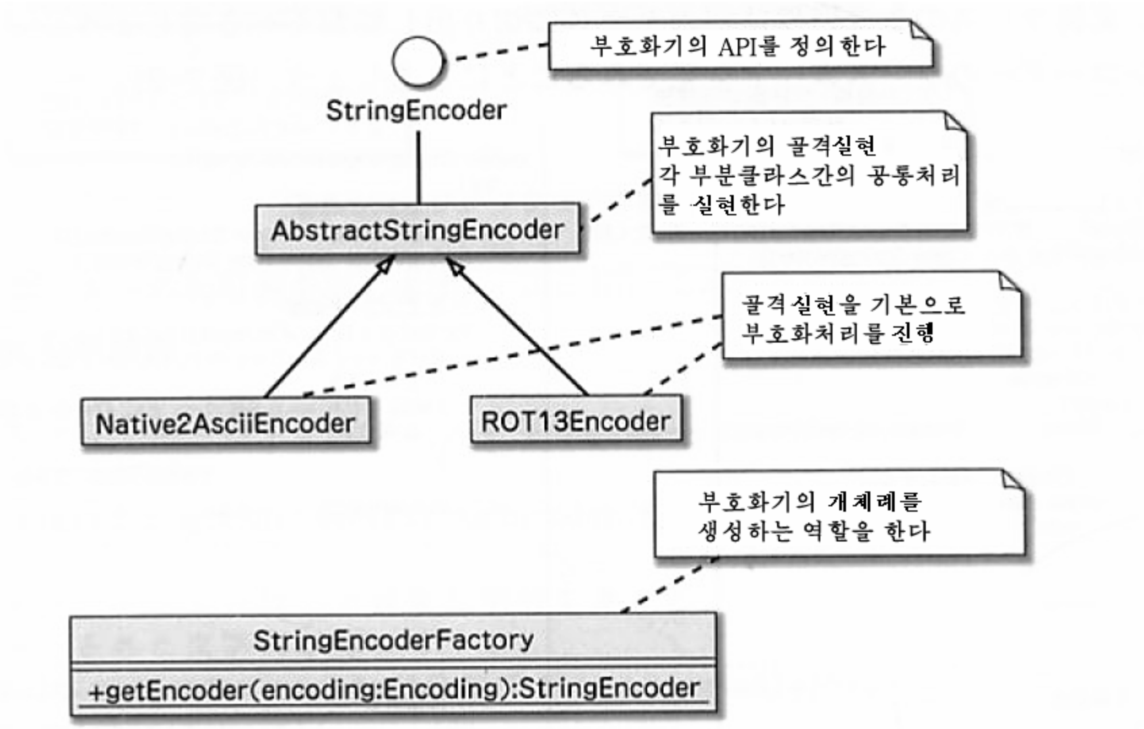


그림 7-3. 개편한 다음의 클라스

어떤 복잡한 개편조작에 대하여서도 그것을 진행한 즉시에 [Refactoring]→[Restore](alt+shift+z)로 조작을 취소시킬수 있다.

7.3.3. encode메쏘드를 형판화

Native2AsciiEncoder의 Encode메쏘드에 대해서 보기로 한다. 이 메쏘드는 준비, 부호화처리, 후처리의 3단계를 거친다.

여기서 부호화처리만은 이 클라스만이 가지는 처리이며 준비와 후처리는 다른 부호화기에서도 공통적인 처리이다. 그리고 이 메쏘드에서 init(준비), doEncode(부호화처리), dispose(후처리)의 3개 메쏘드를 추출하고 encode메쏘드는 그것들을 순서대로 호출하기만 하는 형판메쏘드로 한다. 이때 공통처리부분을 상위클라스에 작성해놓음으로써 중복코드를 쉽게 없앨수 있다. 매 부호화기자체의 처리는 encode()가 호출하는 doEncode()를 덧쓰기(overwrite)하여 실현한다.

그러면 먼저 준비단계로서 처리부분을 선택하여 [Refactoring]→[Extract Methods](Alt+shift+M)을 실행한다(그림 7-4).

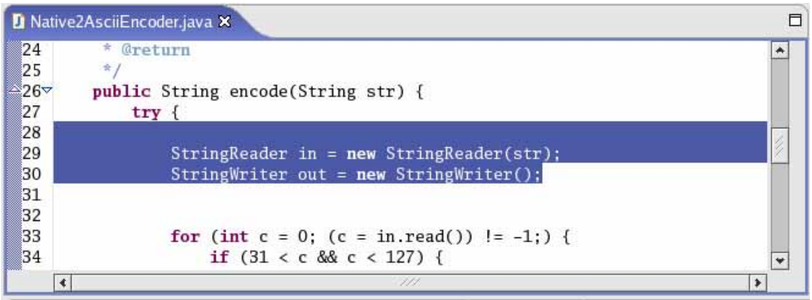


그림 7-4. 추출범위를 선택

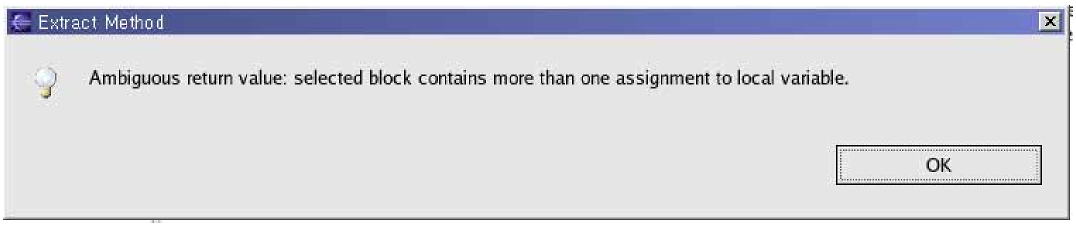


그림 7-5. 메쏘드추출의 실패

실패하면 그림 7-5와 같은 통보창이 나타난다. Java메쏘드에서는 값을 하나밖에 되돌려줄수 없기때문에 국부변수에로의 대입이 2개이상 있는 경우 메쏘드를 추출할수 없다. 이러한 경우 다음과 같은 2가지 대책이 있다.

- 추출범위를 조절하여 국부변수에로의 대입을 한곳에서 진행한다.
- 국부변수를 마당으로 변환한다.

여기서는 두번째 방법을 리용한다. 국부변수 in을 찰칵하여 반전상태로 놓고 [Refactoring]→[Convert Local Variable to Field](Alt+shift+F)을 실행하면 그림 7-6과 같은 대화칸이 현시된다.

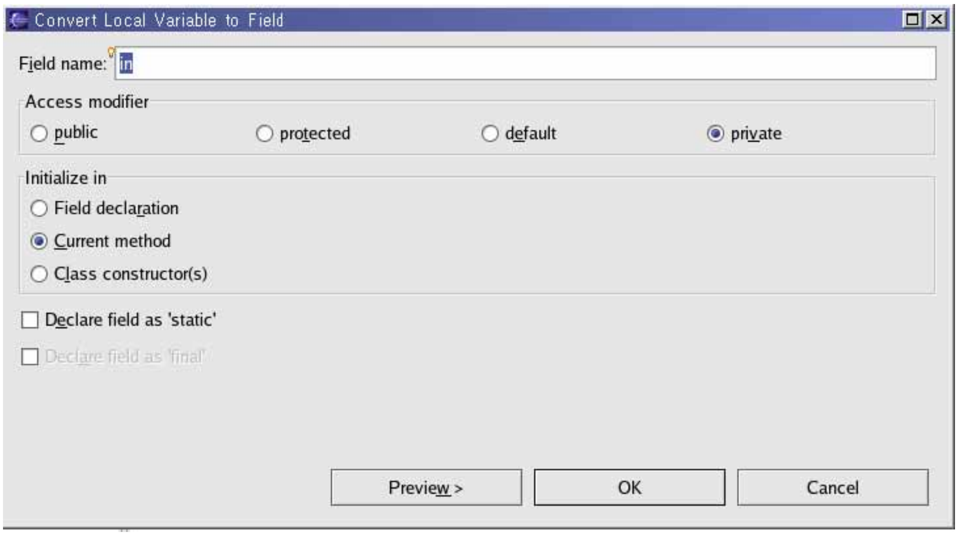


그림 7-6. 국부변수를 마당으로 변환

접근수식자나 초기화의 지정은 기정값으로 설정하고 [Preview]단추를 찰칵한다(그림 7-7).

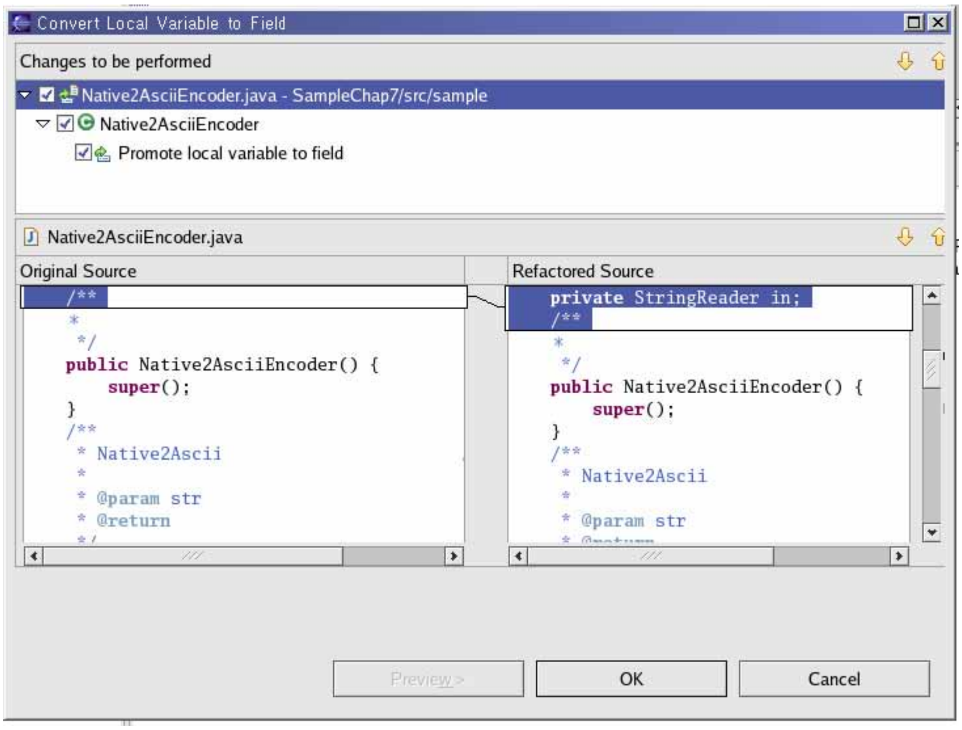


그림 7-7. 미리보기대화칸

여기서 원천코드가 어떻게 변경되였는가를 실행전에 확인할수 있다. 마당이 추가된것을 확인한 다음 [OK]단추를 찰칵한다. 마찬가지로 out도 마당으로 변환한다(그림 7-8).

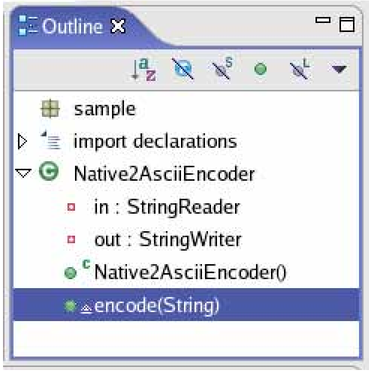


그림 7-8. 마당의 작성

이제는 앞에서 실패했던 [메쏘드의 추출]조작을 다시 한번 시도해본다. 이번에는 정확히 [Extract Methods]대화칸이 현시된다(그림 7-9).

메쏘드이름을 《init》로 한다. [Preview]단추를 찰칵하여 확인한다(그림 7-10).

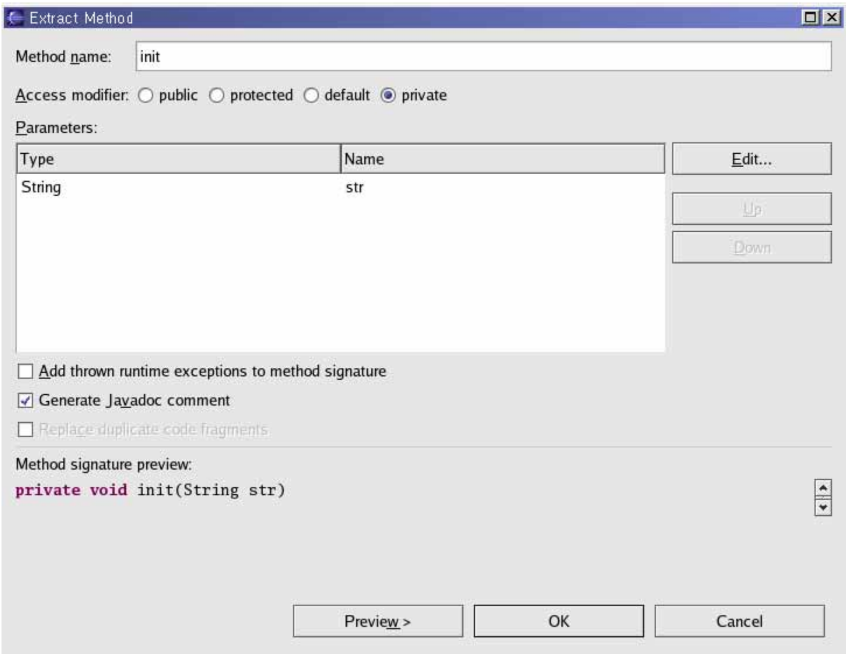


그림 7-9. 메쏘드의 추출

[OK]단추를 찰칵하면 메쏘드가 추출된다.

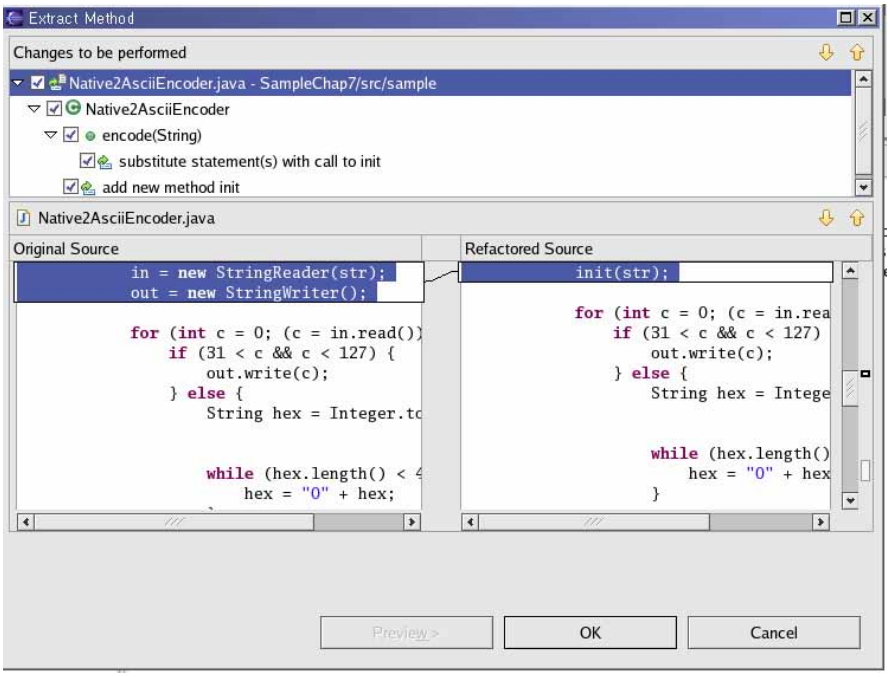


그림 7-10. 메쏘드의 추출 미리보기

다음으로 부호화처리부분도 다른 메쏘드에로 추출된다(그림 7-11).

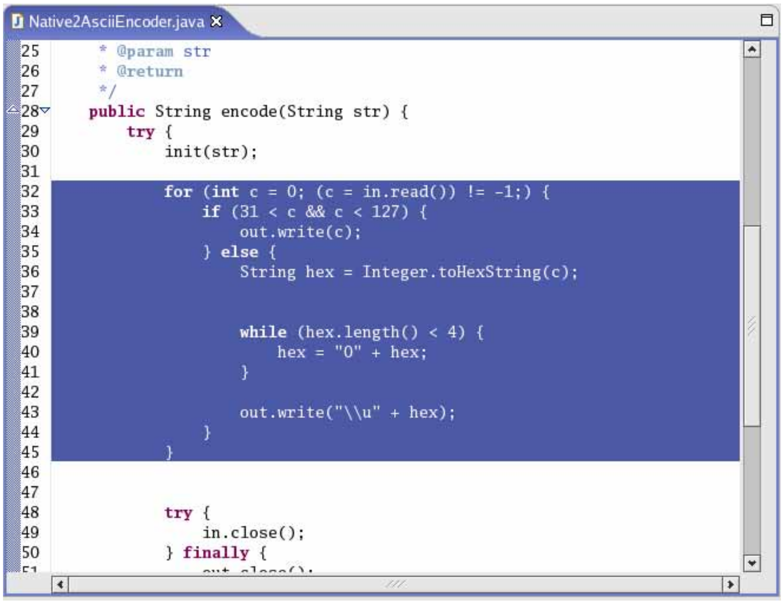


그림 7-11. 부호화처리를 추출

이로부터 추출하는 메쏘드는 매 부호화기(stringEncoder의 보조하위클라스)가 자체의 부호화처리를 실현할수 있도록 재정의되여야 한다. 이와 같은 메쏘드를 《갈구리메쏘드》라고 부르며 doXXX라는 이름을 가지는 경우가 많다.(HttpServlet의 doGet, doPost 등은 그의 전형적인 실례이다.) 여기서는 관습에 따라 이 메쏘드에 《doEncode》라는 이름을 붙인다(그림 7-12).



그림 7-12. doEncode의 추출

마지막으로 후처리부분은 《dispose》라는 메쏘드에로 추출된다.
결과적으로 encode메쏘드는 다음과 같다.

```
/**
*Native2Ascii방식으로 문자렬을 부호화한다.
*
*@param str부호화대상문자렬
*@return 부호화결과
*/
public String encode(String str){
   try{
   init(str);
   doEncode();
   dispose();
   return out.getBuffer().toString();
   }catch(IOException ioe){
          return str;
   }
}
```

이 시점에서 Native2Ascii부호화기는 그림 7-13과 같다.

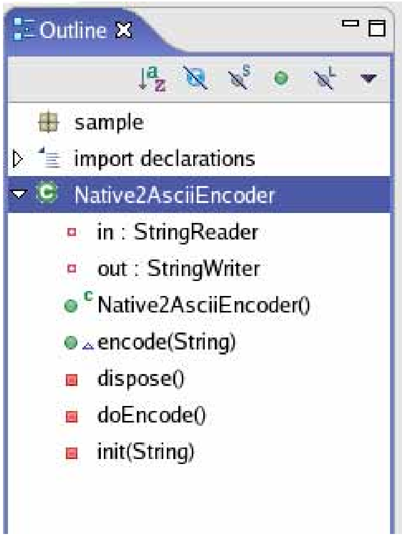


그림 7-13. Native2Ascii부호화기

여기서 메쏘드의 추출이 완료되였다. 그러나 ROT13Encoder와의 중복코드는 아직 해결하지 못했다.

7.3.4. encode메쏘드의 끌어올리기

이제 중복코드의 처리를 진행한다. 순서는 다음과 같다.

① Native2AsciiEncoder의 encode메쏘드 등 모든 공통처리부분을 상위클라스에 끌어올린다. 이때 doEncode메쏘드는 상위클라스에서는 abstract로 선언하고 하위클라스에서는 덧쓰기하면 된다.

② ROT13Encoder의 doEncode메쏘드를 실현한다.

7.3.5. 끌어올리려는 성원의 선택

그러면 먼저 Native2AsciiEncoder의 encode메쏘드를 선택하여 [Refactoring]→[Pull up]을 실행한다(그림 7-14).

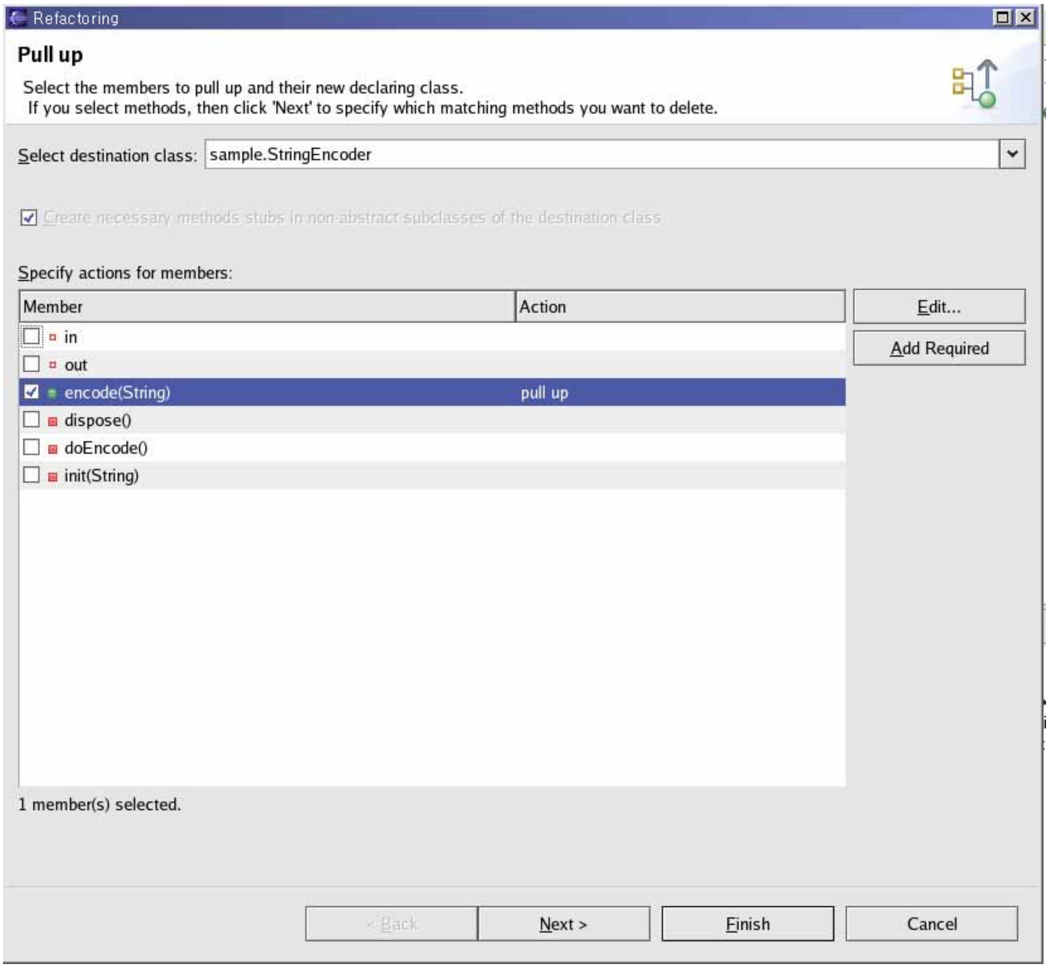


그림 7-14. 끌어올리기

끌어올리려는 성원으로서 encode(String)가 선택되여있지만 여기서 [Add Required] 단추를 찰칵하면 함께 끌어올릴 필요가 있는 성원들을 Eclipse가 검색하여 선택해준다. 여기서는 모든 마당, 메쏘드가 끌어올리기대상으로 되였다.

7.3.6. doEncode를 추상메쏘드로 선언

그러나 doEncode는 Native2AsciiEncoder자체의 알고리듬을 서술한 메쏘드이므로 그대로 상위클라스에 끌어올릴 필요는 없다.

[action]칸을 찰칵하여 [Declare abstract in destination]을 선택한다. 이렇게 하면 doEncode메쏘드는 StringEncoder클라스에서 추상메쏘드로 선언되여 하위클라스에서 덧쓰기할수 있게 된다(그림 7-15).

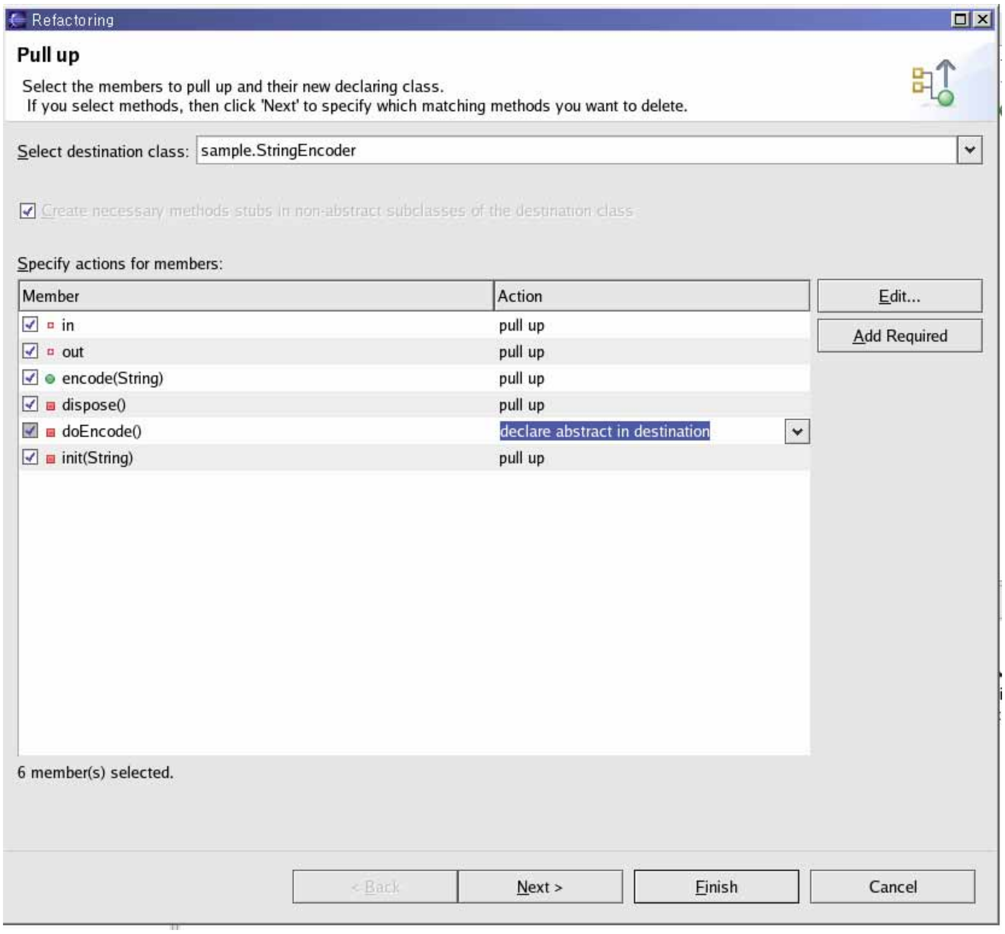


그림 7-15. 끌어올리려는 메쏘드

doEncode메쏘드가 아직 존재하지 않는 ROT13Encoder(부호화기실현클라스)에는 형태만 자동적으로 정의된다. [Next]단추를 찰칵하여 필요없는 메쏘드의 제거작업에 들어간다.

7.3.7. 필요없는 메쏘드의 제거

Native2AsciiEncoder클라스로부터 다음의 성원이 상위클라스에 끌어올려진다.

- 마당 in, out
- 메쏘드 encode, init, dispose

이것들은 부호화기의 종류에는 관계없는 공통적인 처리이므로 하위클라스에는 필요가 없다. 또한 상위클라스에 있는 StringEncoder에는 추상메쏘드 encode가 남아있지만 이것도 Native2AsciiEncoder로부터 끌어올린 encode메쏘드에 의하여 치환되므로

필요없다.

이 대화칸에서 끌어올리기에 의하여 불필요하게 되는 메쏘드들을 선택하여 놓으면 Eclipse가 자동적으로 삭제한다(그림 7-16).

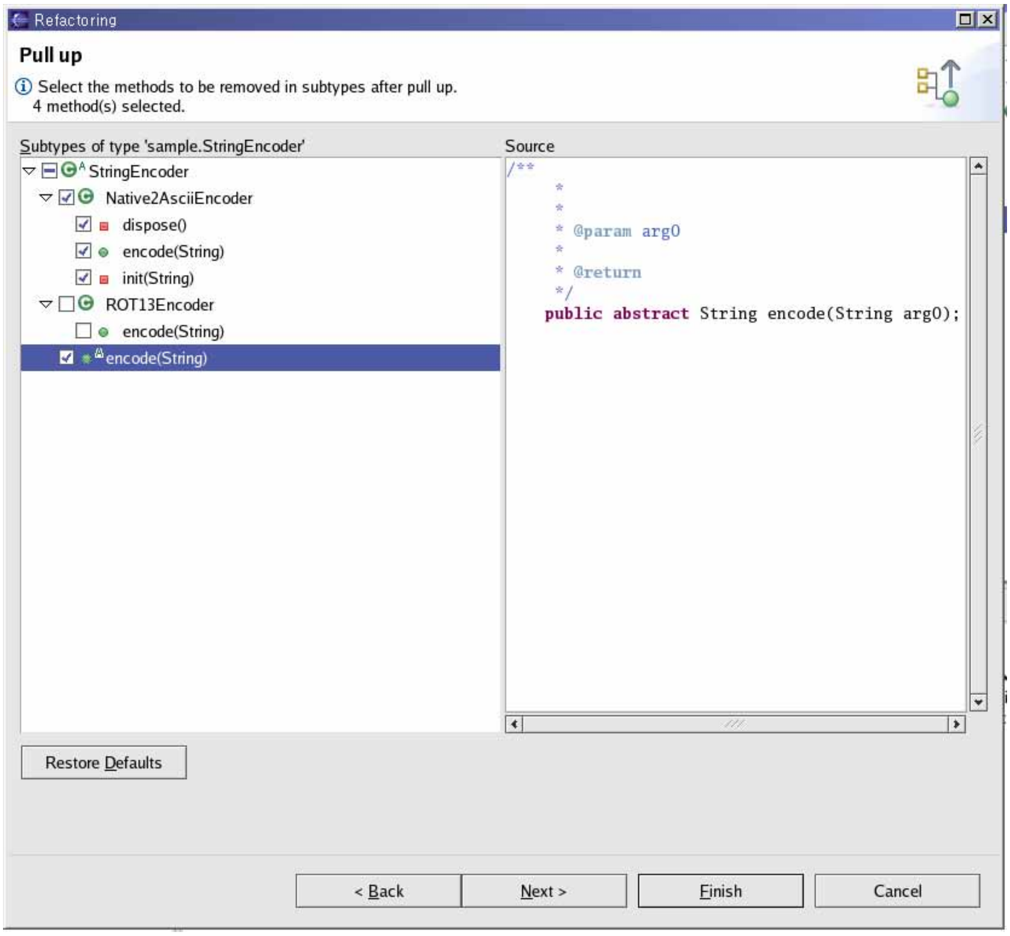


그림 7-16. 불필요한 메쏘드의 제거

7.3.8. 미리보기 및 실행

[Next]단추를 찰칵하여 StringEncoder클라스에 마당이나 메쏘드가 이동되였는가, doEncode메쏘드가 정확히 추상메쏘드로 선언되였는가를 확인한다(그림 7-17). 또한 작성한 단계에서 private된 마당이나 메쏘드 init가 protected로 변경되였다는것을 알수 있는데 이것은 하위클라스로부터 호출되도록 자동적으로 필요한 변경이 진행되였기때문이다.

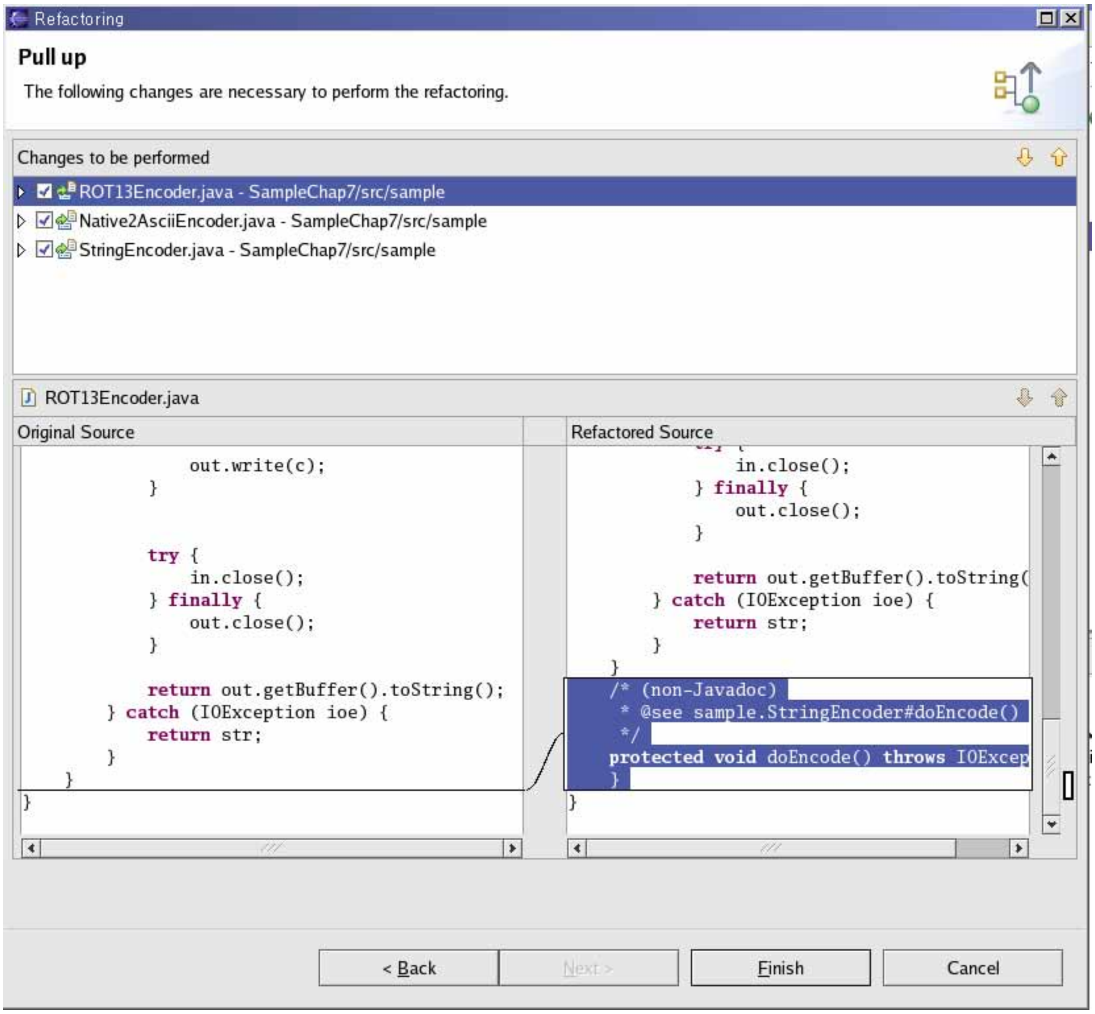


그림 7-17. 끌어올리기의 미리보기

[Finish]단추를 찰칵하면 개편이 완성된다.

7.3.9. 나머지부분의 처리

ROT13Encoder에는 메쏘드추출을 진행하기 전의 encode메쏘드가 남아있다. 이 encode메쏘드로부터 부호화처리부분만을 doEncode메쏘드로 뽑아내야 한다. doEncode메쏘드의 형판이 자동적으로 작성되여있으므로 이것을 리용하여 메쏘드의 추출을 쉽게 진행할수 있다(그림 7-18).

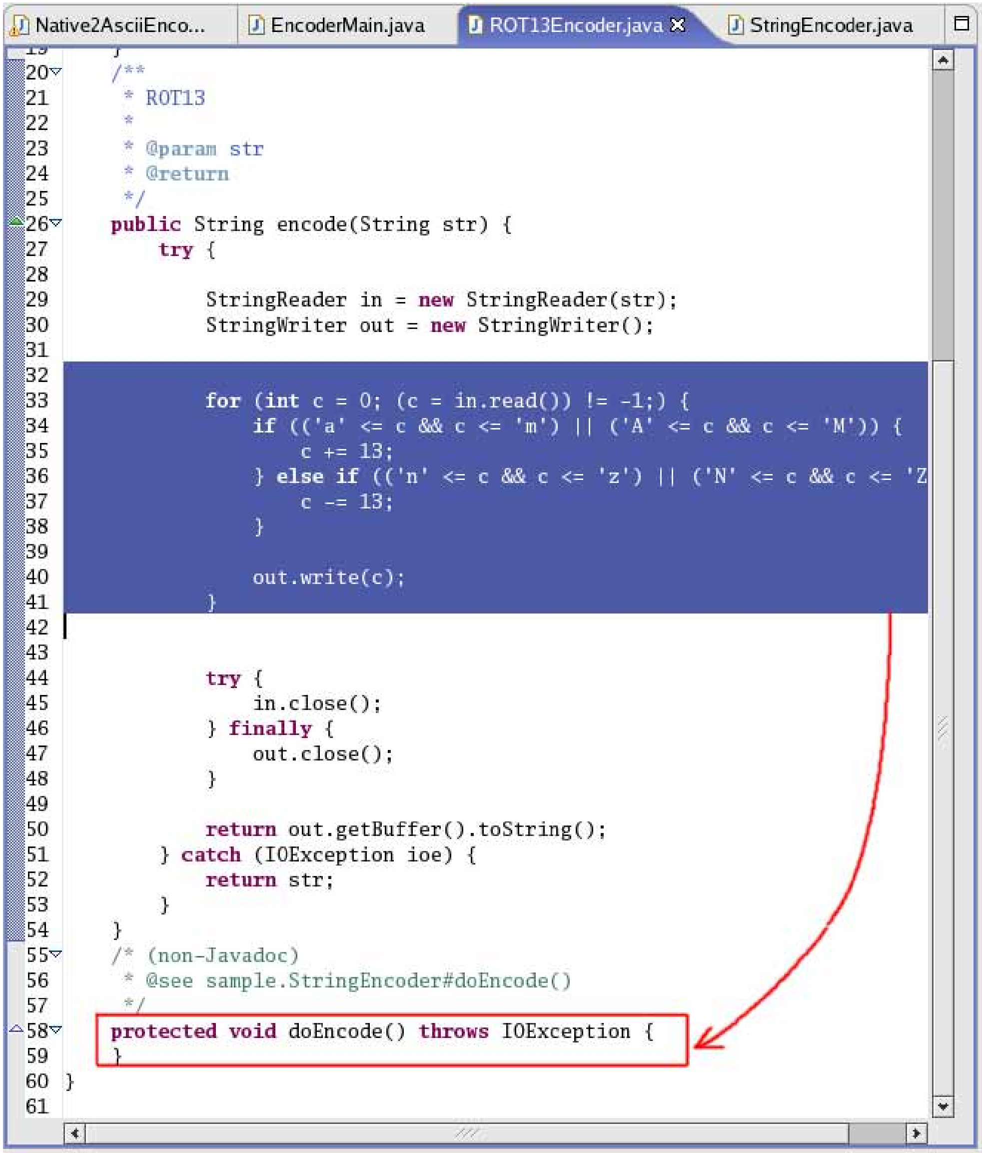


그림 7-18. ROT13Encoder의 doEncode메쏘드를 실현

doEncode메쏘드가 완성되면 이미 있던 encode메쏘드의 잔상을 잊지 말고 제거한다. 이상으로 중복코드가 완전히 없어진다.

다음단계로 들어가기 전에 한번 시험을 진행해본다. [SampleChap7]프로젝트를 선택하여 [Run]→[Run As]→[JUnit Test]을 실행한다(그림 7-19).

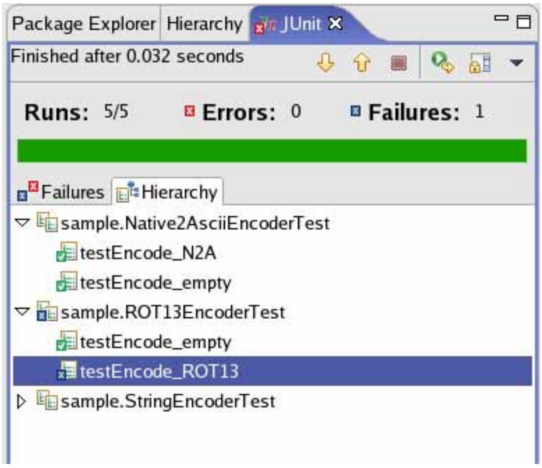


그림 7-19. 푸른색띠가 나타나는 실행대화칸

7.3.10. 구체레생성론리의 분리

이제 다음과 같은 두가지 문제가 남아있다.

- StringEncoder로부터 부호화기의 구체레생성론리를 갈라 낸다.
- 부호화기의 API정의를 대면으로서 추출한다.

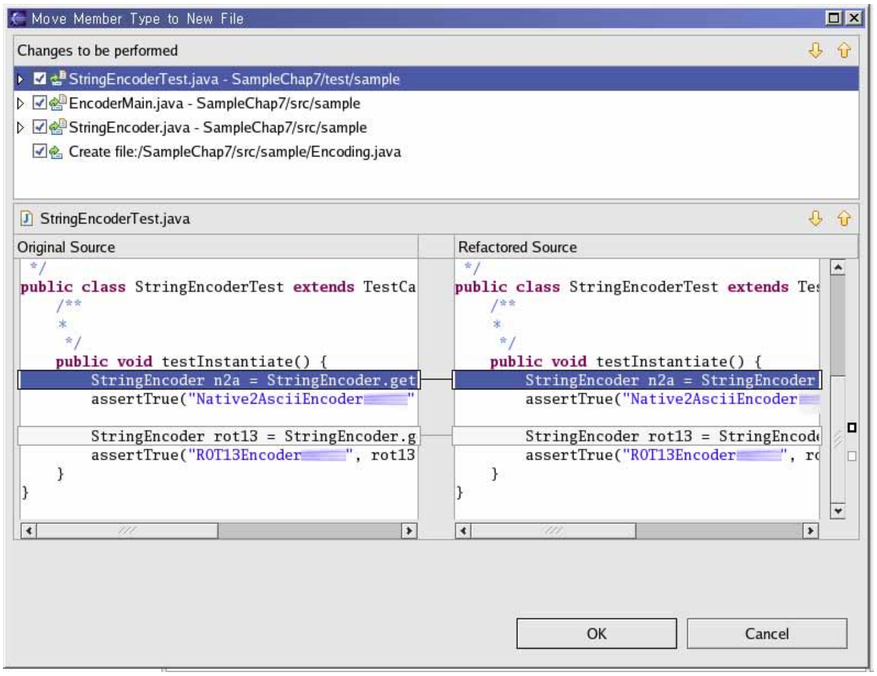


그림 7-20. StringEncoder.Encoding을 상위클라스로 변환

먼저 구체레생성론리의 갈라내기를 진행한다. 순서는 다음과 같다.

① StringEncoder.Encoding을 상위클라스로 변환

② StringEncoderFactory클라스의 새로 작성

③ getEncoder메쏘드를 StringEncoder에서 StringEncoderFactory로 이동

먼저 StringEncoder.Encoding을 선택하여 [Refactoring]→[Move Member Type to New File]을 실행한다(그림 7-20).

[OK]단추를 찰칵한 다음 차림표에서 [File]→[New]→[Class]를 선택하면 새로운 클라스창조대화칸이 나타난다. 여기서 StringEncoderFactory클라스를 작성한다(그림 7-21).

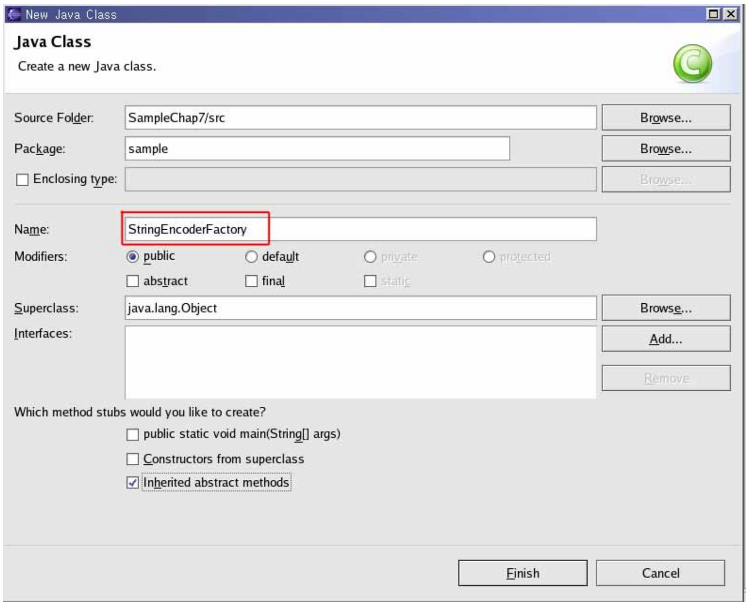


그림 7-21. StringEncoderFactory의 작성

다음으로 StringEncoder클라스의 getEncode메쏘드를 StringEncoderFactory로 이동한다. getEncoder메쏘드를 선택하고 차림표띠에서 [Refactoring]→[Move]을 실행한다(그림 7-22).

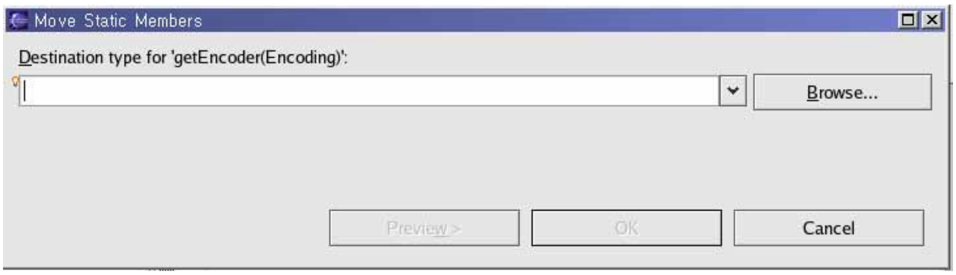


그림 7-22. 성원의 이동

[Browse]단추를 찰칵하여 현시되는 [Choose Type]대화칸에서 메쏘드가 이동되여 갈 클라스를 선택한다(그림 7-23).

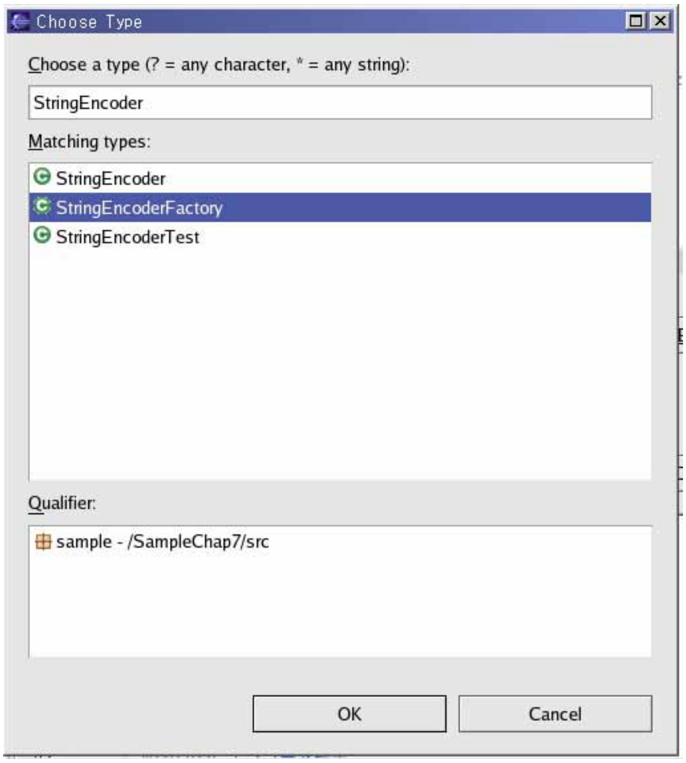


그림 7-23. [Choose Type]대화칸

[OK]단추를 찰칵하면 [Move Static Members]대화칸이 현시되는데 여기서 [Preview]를 찰칵하여 확인한다. 그렇게 하면 그림 7-24와 같은 경고가 나타날수도 있지만 이에 상관없이 [Continue]를 찰칵하면 미리보기화면이 현시된다(그림 7-25). StringEncoder.getEncoder를 호출한 부분이 시험클라스까지 포함하여 모든 StringEncoderFactory의 호출로 바꾸어진다. 참조를 모두 검색하여 남김없이 갱신한다고 하는것이 바로 개편도구의 위력이 있다. [OK]단추를 찰칵하면 메쏘드의 이동이 완료된다.

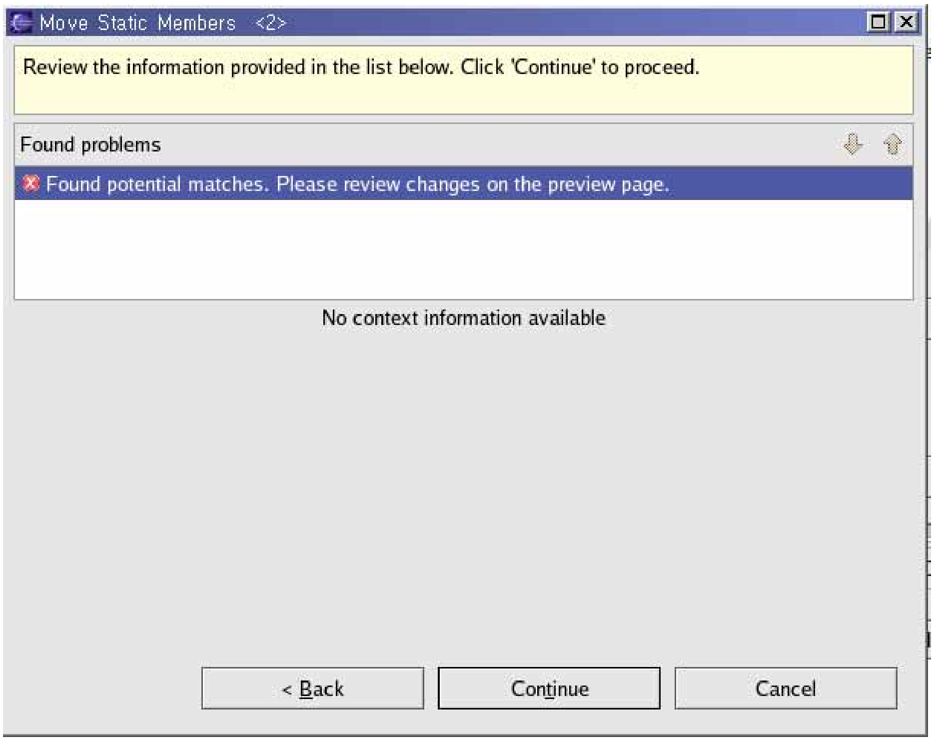


그림 7-24. 경고화면

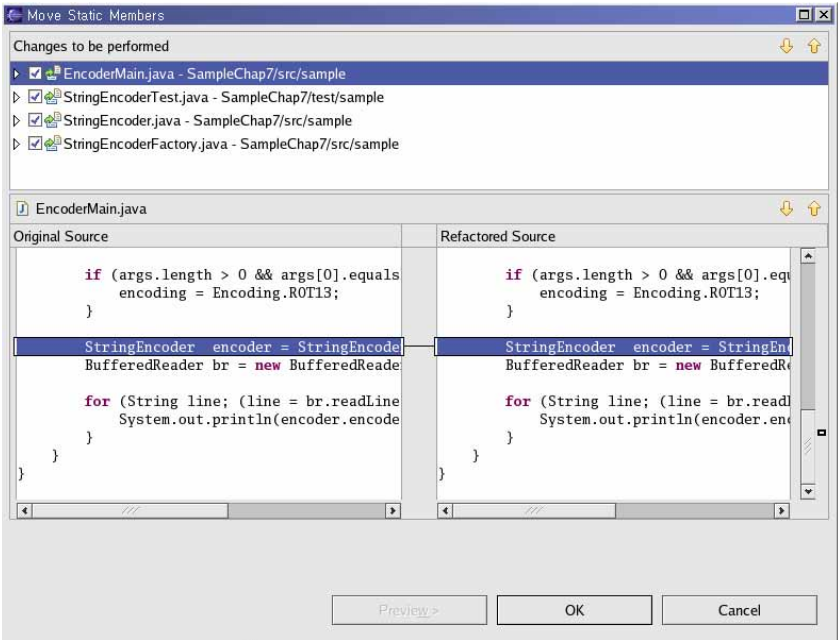


그림 7-25. 메쏘드이동의 미리보기

7.3.11 대면의 추출

마지막으로 남은 문제는 StringEncoder클라스로부터 부호화기의 API정의를 갈라내는 문제이다. StringEncoder클라스는 어디까지나 부호화기의 골격을 실현하지 외부 API를 정의하는것을 목적으로 하지는 않는다. 그래서 이것을 AbstractStringEncoder와 골격에 어울리는 이름으로 변경하여 새롭게 StringEncoder대면을 정의한다.

무엇보다 먼저 이름변경을 진행한다. StringEncoder를 선택한 다음 차림표띠로부터 [Refactoring]→[Rename]을 실행한다(그림 7-26).

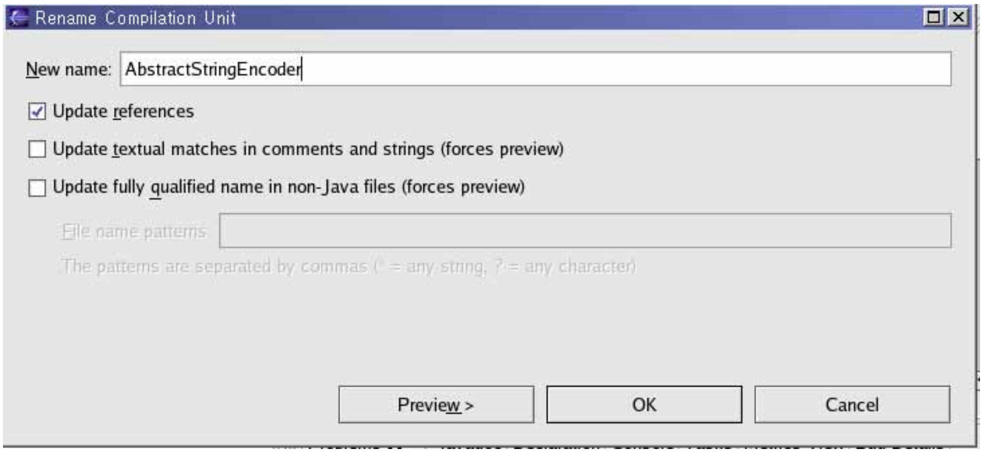


그림 7-26. 클라스의 이름변경

이때 public클라스의 이름변경으로 원천코드안의 참조갱신뿐아니라 파일이름의 변경도 진행해야 하는데 이것을 모두 Eclipse가 한다(그림 7-27). 이렇게 하여 StringEncoder는 골격에 어울리는 이름으로 변경되였다.

다음으로 이 추상클라스에서 부호화기의 API만을 골라낸다. AbstractStringEncoder를 선택한 다음 차림표띠로부터 [Refactoring]→[Extract Interface]을 실행한다. 대면이름을 《StringEncoder》으로 입력하고 대면안에서 선언되는 성원을 선택하여 둔다. 또한 [Change references to the class 'AbstractStringEncoder' into references to the interface (where possible)] 을 선택하면 AbstractStringEncoder에로의 참조가 새롭게 선언되는 대면 StringEncoder를 참조하도록 변경된다(그림 7-28).

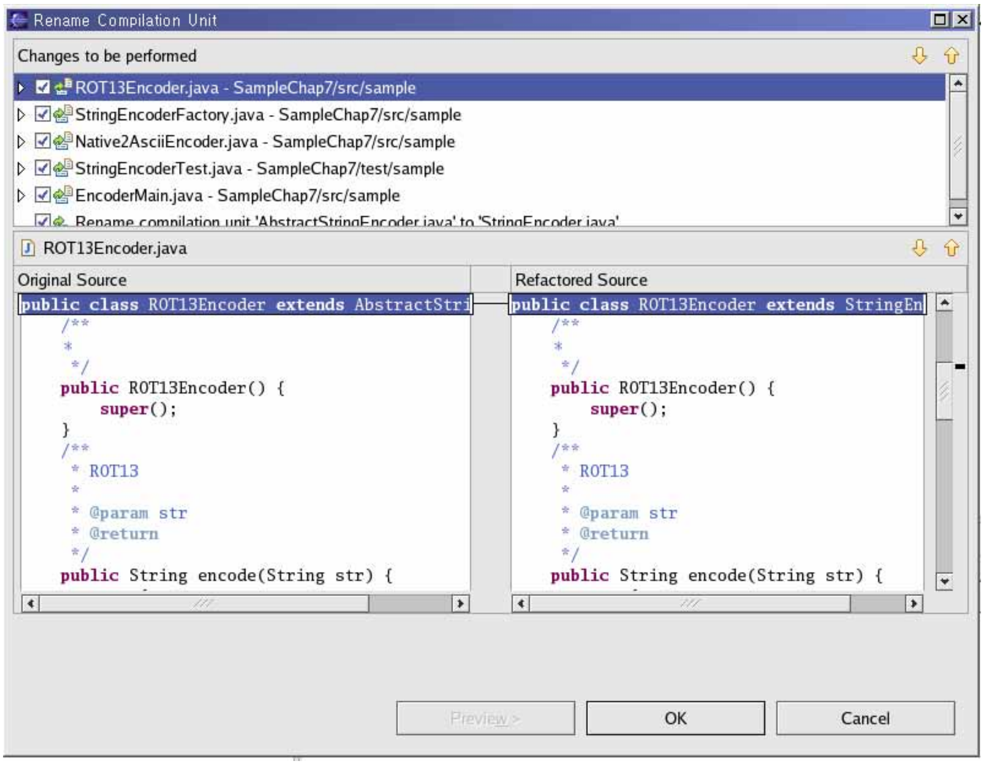


그림 7-27. 클라스이름변경의 미리보기

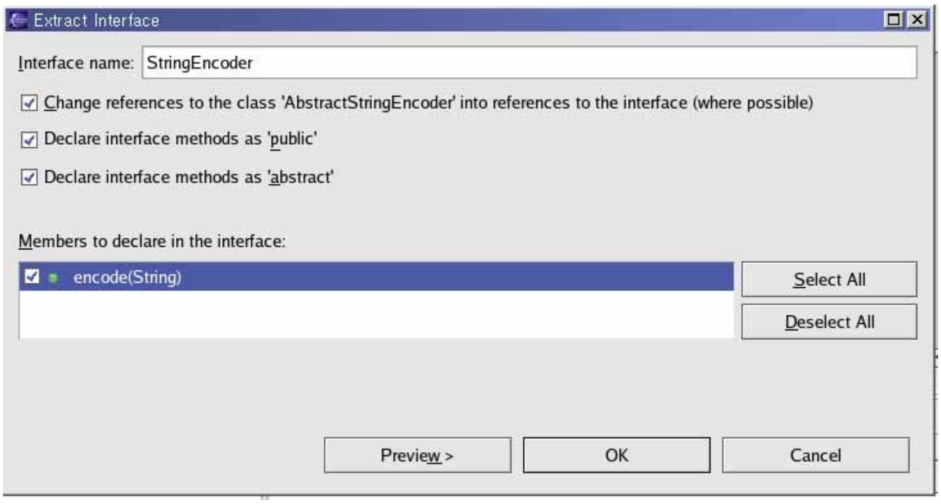


그림 7-28. 대면의 추출

[Preview]단추를 찰칵하면 또 경고가 나타나지만 이에 상관없이 [Continue]를 찰칵한다(그림 7-29).

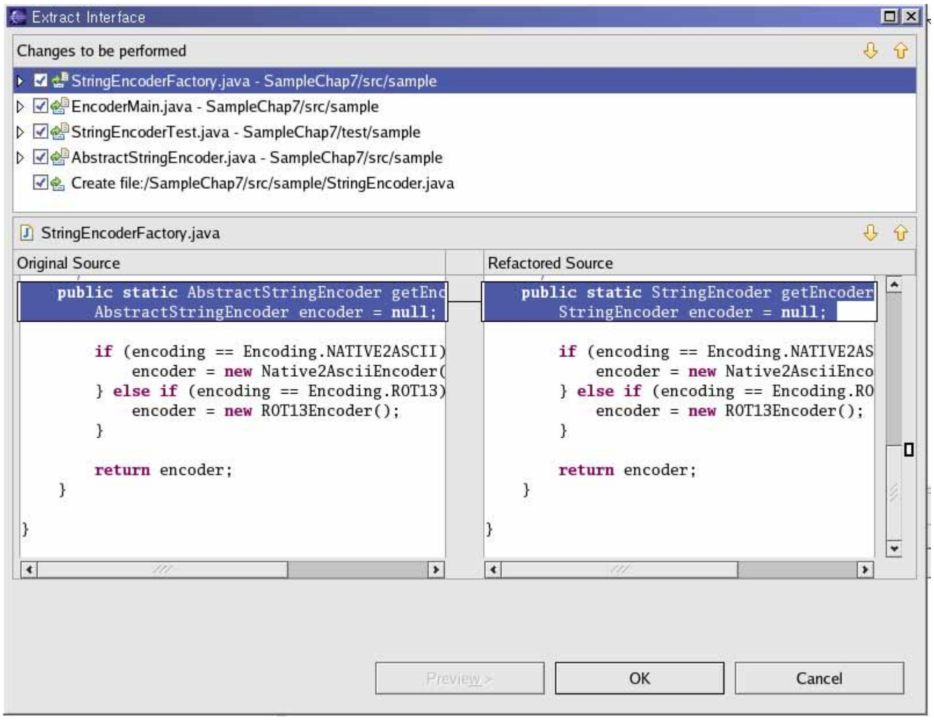


그림 7-29. 대면추출의 미리보기

그림에서 보는바와 같이 AbstractStringEncoder형의 변수가 모두 StringEncoder형으로 변경되였다. 다음 [OK]단추를 찰칵한다. 이것으로서 API정의(StringEncoder), 부호화기의 대면실현(StringEncoderFactory), 부호화기의 골격(AbstractStringEncoder)이 완성되였다.

그러면 마지막으로 한번 더 시험을 진행하여 개편을 끝낸다(그림 7-30).

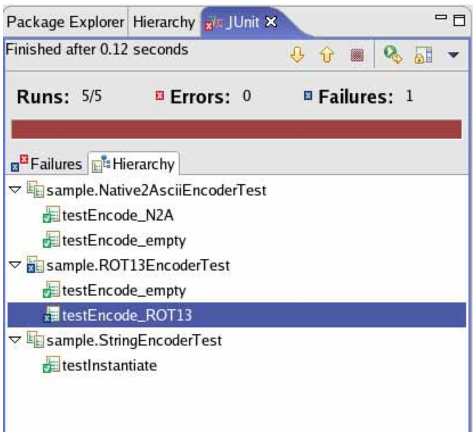
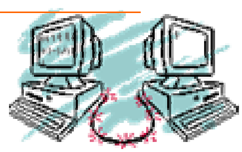


그림 7-30. 완성된 코드



제4절. 기타 개편조작

지금까지 다음과 같은 개편조작에 대하여 소개하였다.

- 이름변경
- 이동
- 메쏘드의 추출
- 끌어올리기
- 국부변수를 마당으로 변환
- 대면의 추출
- 겹싼형의 상위형에로 변환

이외에도 몇가지 사용빈도가 높은 조작방법이 있는데 여기서 종합적으로 설명하기로 한다.

7.4.1. 제작소(factory)의 도입

구성자를 public로 선언하면 그의 클라스의 보수와 확장이 가능하다. 실례로 여러 사람들이 오래동안 사용하고있는 틀거리(framework)의 보수를 진행한다고 생각해보자.

먼저 어떤 클라스에 public구성자를 정의한다. 틀거리의 사용자는 이 구성자를 사용하여 구체레를 생성하는 코드를 작성하게 된다. 그런데 마침내 그는 《이 클라스는 하나로 할수 있었다. 구체레를 대량생산할 필요는 없다.》는것을 알게 될것이다. 새로 단일구체레를 되돌려주는 제작소메쏘드를 정의할수는 있지만 public구성자를 사용하고있는 사용자가 대부분인 이상 이제 와서 구성자를 private로 바꿀수는 없다. 그리하여 《제일 처음부터 제작소메쏘드를 정의하여 놓았으면 좋았겠는데》하고 후회하게 된다.

《제작소의 도입》은 이러한 페단을 미연에 방지하기 위하여 public구성자를 private로 변환하여 구체레생성용의 제작소메쏘드를 정의하는 개편방법이다.

여기서는 실례로서 Native2AsciiEncoder클라스의 구성자에서 제작소메쏘드를 생성하기로 한다. 패키지열람기에서 Native2AsciiEncoder클라스의 구성자를 선택하고 [Refactoring]→[Interface Factory]을 실행한다. 그러면 [Interface Factory]대화칸이 현시된다(그림 7-31).

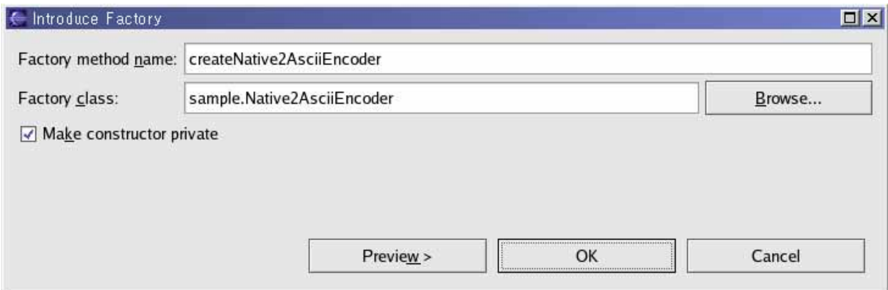


그림 7-31. 제작소에 관한 설정대화칸

또한 제작소메쏘드를 Native2AsciiEncoder클라스가 아니라 다른 클라스로 만들수도 있다. 이 실례에서는 기정으로 설정한다. [Preview]를 찰칵하면 그림 7-32와 같은 대화칸이 현시된다.

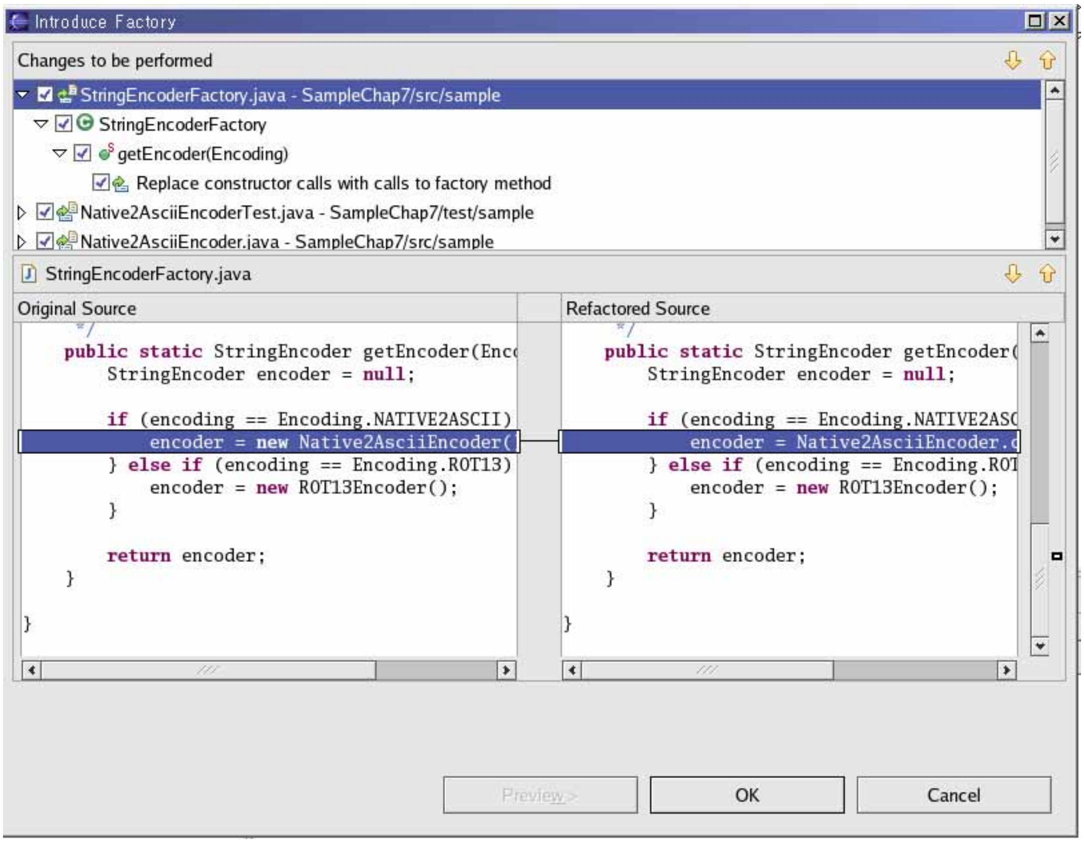


그림 7-32. 제작소도입의 미리보기대화칸

구성자가 private로 되고 제작소메쏘드가 생성되며 구성자를 호출하는 다른 클라스도 제작소메쏘드를 사용하도록 변경된다.

7.4.2. 상수의 추출

이것은 기계어코드로 된 수값이나 문자렬을 static final의 마당으로 선언하고 그의 참조로 치환하는 조작이다. 의미가 명확치 않은 수자는 이 개편을 적용하여 리해하기 쉬운 상수로 치환하기도 한다.

여기서는 실례로서 앞의 ROT13Encoder클라스의 doEncode메쏘드에 있는 《13》이라는 수자를 상수로 치환한다(그림 7-33).

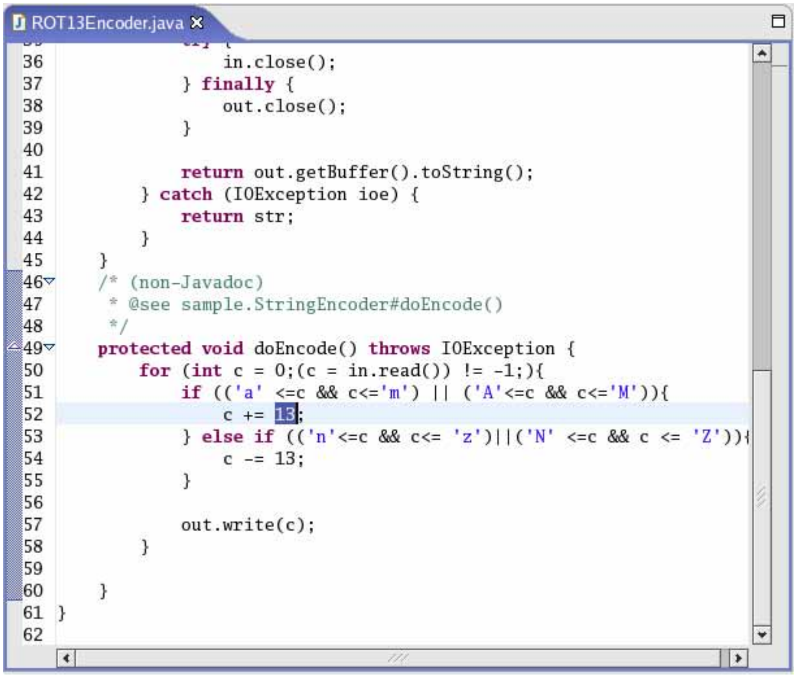


그림 7-33. 수자《13》의 추출

《13》을 두번 찰칵하여 선택하고 [Refactoring]→[Extract Constant]을 실행한다(그림 7-34). [constant name]에 《SHIFT_WIDTH》를 입력한다. [Replace all occurrences of the selected expression with references to the constant]검사칸을 선택하고 [Preview]단추를 찰칵한다(그림 7-35).

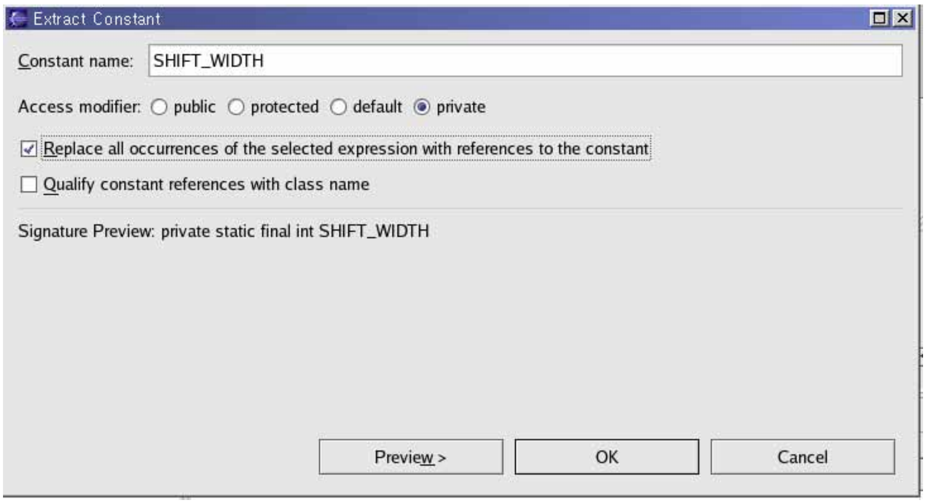


그림 7-34. 추출대화칸

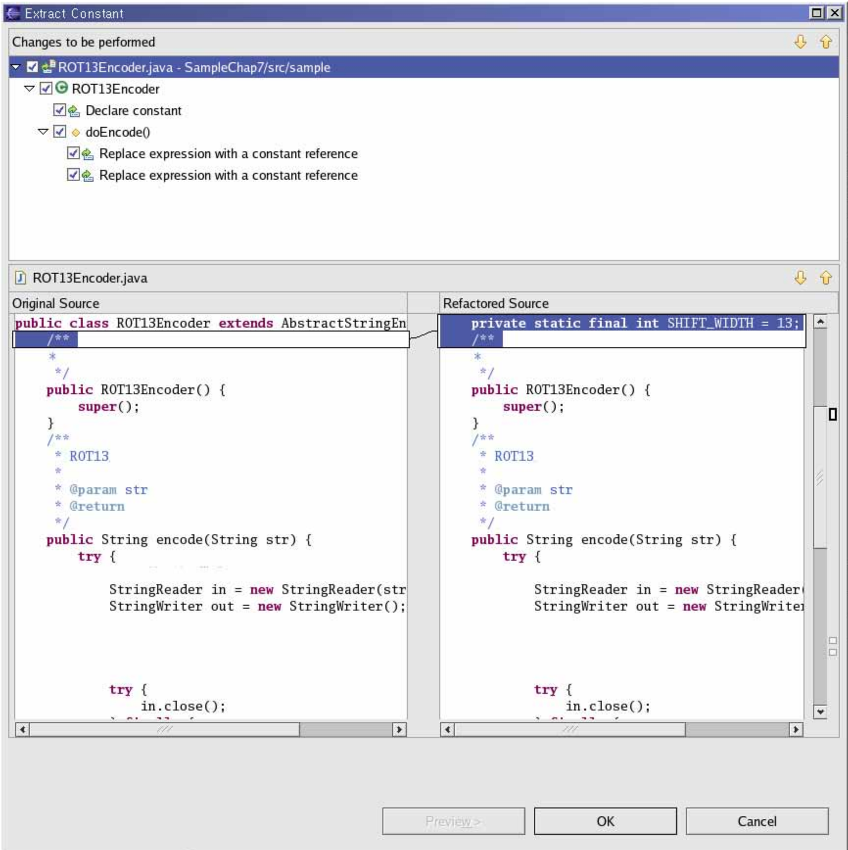


그림 7-35. 추출결과

이 실례에서는 추출대상이 불과 2개 밖에 없었지만 이것이 몇백, 몇천개 있어도 Eclipse는 하나도 빠짐없이 완벽하게 수정을 진행한다. 역시 같은 조작으로 국부변수의 추출도 할수 있다.

7.4.3. 제자리화

상수의 추출, 국부변수의 추출은 메쏘드의 추출과 같이 《추출계조작》이라고 불리운다.

여기서 설명하는 제자리화(in line)는 추출계조작과 완전히 반대의 동작을 진행한다. 즉 메쏘드호출이나 상수, 국부변수의 참조를 제거하는 조작이다.

개편에 역행하는것과 같은 이 조작들이 과연 어떤 때에 필요하겠는가. 개편을 반복하여나가면 처음에는 복잡한 메쏘드가 간단하게 되며 종당에는 메쏘드로 만들 여지가 없는 단순한 코드로 되는것도 있다. 그러한 경우에 메쏘드를 제자리화하는것이다. 상수나 국부변수에 관하여도 같다. 여기저기서 참조된 변수가 종당에 한곳에서만 참조되게 될 때 제자리화를 검토한다.

여기서는 앞의 개편에서 제일 처음으로 메쏘드추출을 진행한 AbstractStringEncoder의 init메쏘드를 제자리화하여 본다. Outline보기에서 init메쏘

드를 선택한 다음 차림표띠에서 [Refactoring]→[inline화](Alt+Shift+L)을 실행한다(그림 7-36).

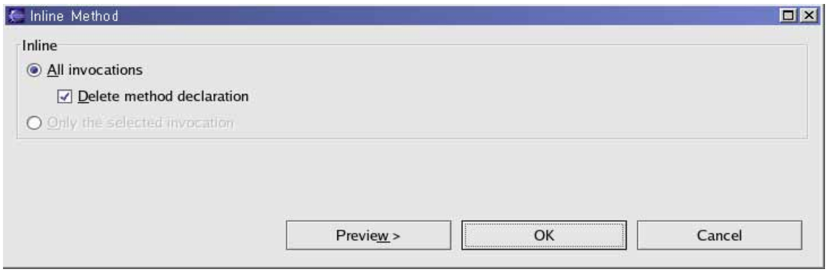


그림 7-36. 제자리화 설정대화칸

init메쏘드를 호출하는 부분을 모두 치환하겠는가 혹은 선택한 부분만을 치환하겠는가를 여기서 설정한다. 이 실례에서는 메쏘드의 호출위치가 아니라 메쏘드선언 그자체를 선택하기때문에 [All invocations]칸을 선택한다. [Delete Method Declaration]를 선택하면 원래의 메쏘드는 삭제된다. [Preview]단추를 찰칵한다(그림 7-37).

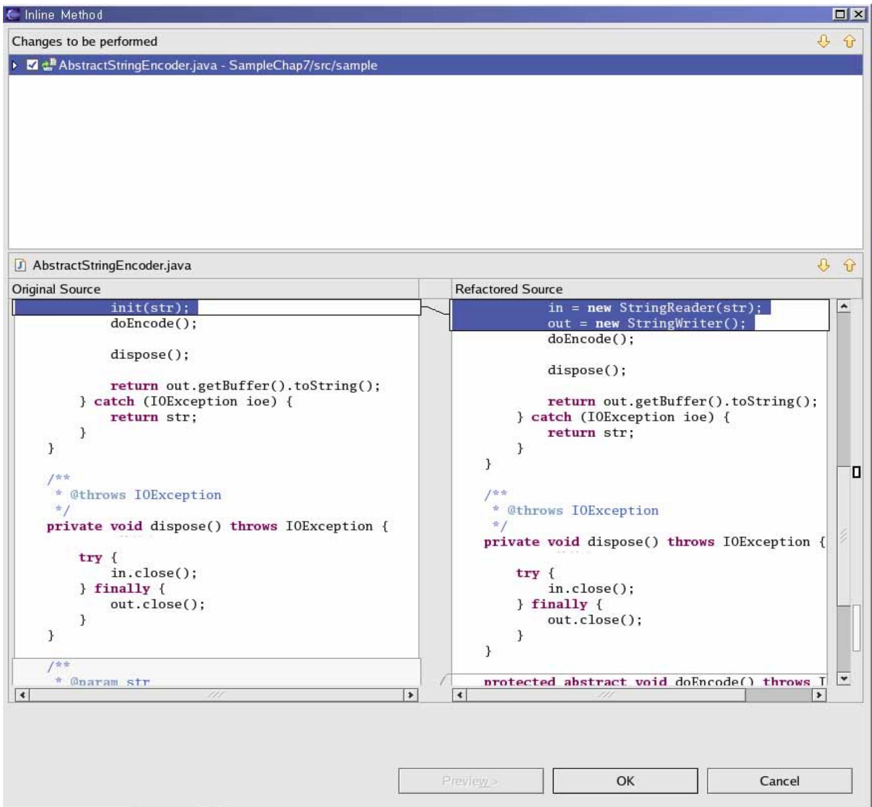


그림 7-37. 메쏘드의 제자리화 미리보기

init메쏘드가 정확히 제자리화되여 《메쏘드의 추출》을 진행하기 전의 상태로 된다. 상수나 국부변수도 같은 방법으로 제자리화할수 있다.

7.4.4. 마당의 은폐화

《마당은 private로 선언하고 Getter/Setter를 통하여 호출해야 한다》는것이 중요한 객체지향프로그람수법의 하나이다. 따라서 마당에 대한 조작을 직접 의뢰자에 맡기는것은 클라스의 은페화를 파괴하는것으로 된다.

[Encapsulate Field]는 마당을 private로 변경하고 Getter/Setter를 자동생성하는 개편이다. 실례로 앞의 AbstractStringEncoder의 《in》마당을 은페시킨다. Outline보기에서 마당 《in》을 선택하고 [Refactoring]→[Encapsulate Field]를 실행한다(그림 7-38).

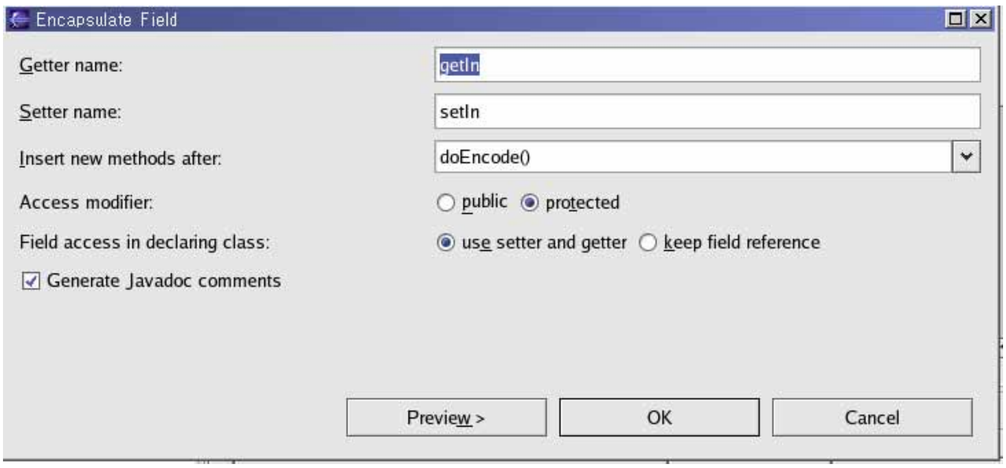


그림 7-38. 마당의 은페화설정대화칸

Getter, Setter의 이름, 접근식별자만이 아니라 원천코드의 어느 위치에 들어있는가 하는 지정까지 진행한다. [Field access in declaring class]에서는 마당이 선언되여있는 클라스내에서 그 마당을 호출할 때 Getter/Setter를 사용하겠는가 혹은 직접 호출하겠는가를 설정한다. 여기서는 대부분의 경우 직접호출을 선택한다. [OK]단추를 찰칵하여 완료한다. 결과 마당 《in》이 private로 되고 호출메쏘드가 생성되였다(그림 7-39).

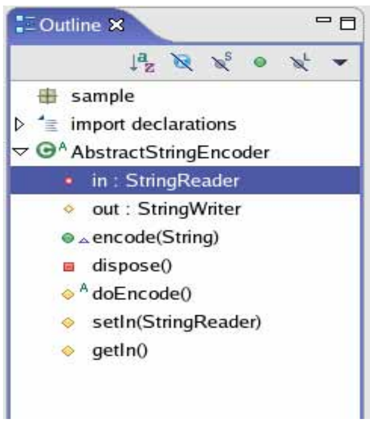


그림 7-39. 마당의 은페화

제8장. Ant지원

제1절. Ant란 무엇인가?

Ant란 ASF(Apache Software Foundation)의 Apache Ant Project (http://ant.apache.org)에서 개발하고있는 쏘프트웨어개발지원도구이다.

주로 Java개발자들을 위해 제작되였고 원천파일의 콤파일, API문서의 생성, jar파일작성 등의 작업을 지령조작으로 간단히 진행할수 있도록 되여있다. 많은 량의 원천파일을 대상으로 일괄처리를 할수도 있고 간단한 정형처리를 자동화할수도 있는 등 통합개발환경(IDE)에서 하기 어려운 작업을 이 도구를 리용하여 수행할수 있다.

또한 Ant는 그 자체가 Java로 서술되였기때문에 실행기반에는 무관계하게 동작한다는것도 이 도구의 특징중의 하나이다.

제2절. 건립파일

Ant는 건립파일을 읽어들여서 동작한다.

건립파일은 XML형식으로 서술되여있는 일종의 스크립트파일로서 Ant가 리해할수 있는 명령들을 조합시켜 여러가지의 처리들을 서술한것이다. Ant는 건립파일을 해석하고 실행시키는 해석기라고도 말할수 있다.

건립파일은 원천파일의 콤파일, JUnit시험클라스의 실행, API문서의 작성, 콤파일된 클라스파일로부터 jar파일의 생성과 같은 4가지 기능을 갖추고있다. 이 파일의 이름을 build.xml이라고 달았는데 원래 Ant의 기정건립파일이름이다.

목록 8-1. build.xml

```
<?xml version= “1.0” encoding= “UTF-8” ?>
<project name= “Refactoring” basedir= “.” default= “build” >
   <description>건립,시험,API문서의 작성,JAR파일의 생성 등을 진행한다.
   </description>

   <!-- 속성 정의 -->
   <property name= “dir.bin”        value= “bin” />
   <property name= “dir.src.main”        value= “src” />
   <property name= “dir.src.test”  value= “test” />
   <property name= “dir.java.doc” value= “docs/api” />
   <property name= “file.jar”       value= “classes.jar” />
```

```
<!-- 목표 정의 -->
<target name= “build” description= “원천파일을 번역한다.” >
<mkdir dir= “${dir.bin}” />
<javac desdir= “${dir.bin}” debug= “false” >
    <src path= “${dir.src.main}” />
    <src path== “${dir.src.test}” />
</javac>
</target>

<target name= “test” depends= “build”
          description= “시험을 진행한다.” >
       <junit printsummary= “yes” >
              <classpath location= “${dir.bin}” />
              <batchtest>
                     <fileset dir= “${dir.src.test}” >
                          <include name= “**/*Test.java” />
                     </fileset>
              </batchtest>
       </junit>
</target>

<target name= “javadoc” description= “API문서를 작성한다. ” >
       <javadoc
           access= “package”
           destdir= “${dir.javadoc}”
           sourcepath= “${dir.src.main}”
           packagenames= “sample” />
</target>

<target name= “archive” depends= “test” description= “JAR파일을 생성한다. ” >
       <jar jarfile= “${file.jar}” >
              <fileset dir= “${dir.bin}” >
                     <exclude name= “**/*Test.class” />
              </fileset>
       </jar>
</target>
</project>
```

제3절. 프로젝트, 목표

건립파일의 제일 바깥요소는 <project>이다. name속성에서 프로젝트의 이름을 지정한다. 실제로는 간략기능이 있지만 이름없이 사용하는것은 불편하기때문에 목록 8-1에서는 프로젝트의 이름을 Refactoring이라고 달았다.

프로젝트의 가운데에 있는 <target>요소에 의하여 목표를 정의한다.

목표는 항상 이름으로 식별되기때문에 <project>의 경우와는 달리 이것들의 속성은 반드시 설정해주어야 한다. Ant를 기동시킬 때에 목표이름을 지정하는것으로서 수행해야 할 동작을 실현할수 있다.

실례로 《build》라고 지정하면 원천파일이 콤파일되고 class파일이 생성된다. 그리고 목표이름을 특별히 지정하지 않고 Ant가 기동하는 경우엔 기정으로 정해진 목표가 실행된다.

기정목표는 <project>요소의 default속성으로 반드시 지정되여야 한다.

8.3.1. 목표의 의존관계

목록 8-1의 test목표를 보면 JUnit의 시험클라스를 모두 추출하여 시험을 진행하는것이 목적인데 그것을 실현하자면 프로그람이 반드시 콤파일되여있어야 한다. 결국 test목표를 실행하기 전에 먼저 build목표가 완료되여야 한다. 이것을 《test는 build에 의존하고있다.》라고 한다.

8.3.2. 과제

이것은 등록부의 작성, 원천파일의 콤파일, jar파일의 작성에 리용한다.

Windows의 지령묶음파일(bat파일)이나 Unix의 쉘스크립트에서는 동작을 조작체계상에서 지령으로 실현하고있다. 그러나 Ant에서는 이러한 기본동작의 하나하나가《과제》로서 정의되여있고 건립파일은 지령이 아닌 과제의 호출에 대하여 서술하고있다.

목록 8-1에서는 <mkdir>, <javac>, <jar>라는 과제가 사용되고있다. 이처럼 지령호출을 과제라는 형식으로 추상화했으므로 Ant의 건립파일은 실행기반에 의존하지 않는다. 목록 8-1의 건립파일을 Windows XP상에서 개발했지만 Unix상에서도 꼭같이 동작시킬수 있다. 조작체계에 의존하는 지령을 실현하려면 <exec>과제를 사용하면 된다.

8.3.3. 속성

원천파일의 위치나 API문서의 출력위치, JAR파일의 이름 등 건립파일에서 자주 사용되는 문자렬을 속성으로 정의해놓으면 편리하다.

<property name= “속성이름” value= “속성값” />

또한 파일속성을 리용하여 java.util.Properties형식의 속성파일을 읽어들일수도 있다.

<property file= “파일이름” />

정의한 속성값을 참조할 때에는 아래와 같이 서술한다.

${속성이름}

<property>요소는 다른 과제들과는 달리 <project>요소의 바로 아래에 놓인다. 이렇게 건립파일전체에 대한 유효속성을 정의할수 있다.

8.3.4. 조합속성

Ant는 표 8-1에 서술되여있는 조합속성을 제공하고있다.

이것들은 <property>요소에서 정의하지 않아도 항상 참조할수 있다.

표 8-1. 조합속성

속성이름	설 명
basedir	기초등록부이다. 건립파일전체에서 상대경로지정은 이 기초등록부를 기준으로하여 진행된다.
ant.file	건립파일의 전체경로이름
ant.project.name	Ant프로젝트이름(Refactoring)
ant.home	Ant의 설치등록부
ant.version	Ant의 판본(2004년 7월 16일에 개발된 Ant의 판본은 1.6.2이다.)
ant.java.version	Ant실행에 사용된 JVM의 판본(1.6)

8.3.5. 체계속성

Java의 체계속성도 Ant의 속성과 같이 참조할수 있다.

아래의 건립파일을 실행하면

```
<project default= “getProperty” >
   <target name= “getProperty” >
          <echo message= “${java.vendor}” />
   </target>
</project>
```

다음과 같이 출력된다.

[echo] Sun Microsystems Inc.

8.3.6. 환경변수

건립파일안에서는 환경변수도 참조할수 있다.

<property>요소의 environment속성에서 환경변수의 앞붙이를 정의할수 있다. 그것은 ${앞붙이.환경변수이름}와 같은 형식으로 참조할수 있다.

아래의 건립파일을 실행하면

```
<project default= “getenv” >
<property environment= “env” />
   <target name= “getenv” >
          <echo message= “${env.SystemRoot}” />
   </target>
</project>
```

다음과 같이 출력된다.

```
[echo] /root
```

이밖에도 많은 기능이 있지만 여기서는 우에서 서술한것만 소개한다. 자세한것을 알려면 다음의 거점을 참조하면 된다.

http://ant.apache.org/manual/index.html

Ant와 Make

Ant는《Java로 된 Make와 같은 건설도구이다.》라고도 불리운다. 건립파일은 역할로 놓고보면 Makefile과 비슷하고 목표라고 하는것도 Makefile안에 꼭같이 정의되여있다. Make를 알고있는 사람들은 《Ant란 Java로 된 Make이다. Makefile대신에 build.xml이라는 XML파일을 해석한다.》라고 하면 Ant의 개요를 파악할수 있을것이다.

그러나 Ant에는 Make와 다른점이 있다.

- Java개발자지향이다.

Make는 주로 C/C++/Fortran언어의 개발작업용으로 리용되지만 Ant는 Java개발작업을 기본으로 하고있다.

- 건립파일을 XML형식으로 서술한다.

Makefile은 자체의 문법을 가지고있고 Make를 사용한 경우는 그에 따라서 필요하지만 Ant의 건립파일은 일반적인 Xml파일이기때문에 습득하기가 쉽고 일반적인 XML편집도구를 사용할수 있다.

들여맞추기(indent)는 반드시 TAB로 진행하여야 한다는 제한도 없다.

제4절. 환경설정

여기에서 실제로 Eclipse의 Ant지원기능을 시험해보기로 한다.

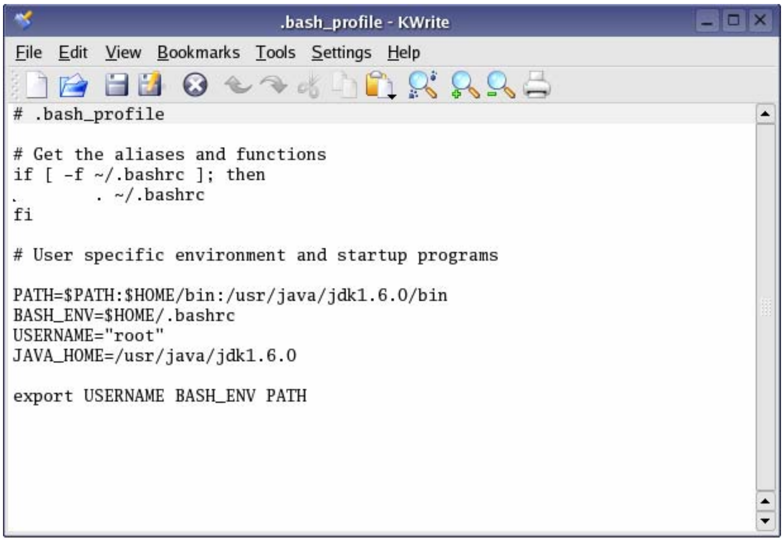
7장에서 사용한 SampleChap7프로젝트에 있는 build.xml을 리용한다.

8.4.1. 환경변수 JAVA_HOME

<javac>나 <javadoc>라는 과제를 실행할 때에는 Ant가 인식하는 클라스경로에 J2SDK의 tools.jar가 포함되여있어야 한다. 이 설정은 후에 설명하는 [Ant실행시간설정]에서 해도 좋지만 환경변수 JAVA_HOME으로 J2SDK(java쏘프트웨어개발도구:Java Software Development Kit)의 설치등록부를 지정해놓으면 Eclipse가 자동적으로 tools.jar를 클라스경로에 추가해준다. JAVA_HOME은 Eclipse뿐만아니라 그외 여러가지 개발도구가 참조하는 표준적인 환경변수이다(그림 8-1).

이 환경변수를 설정하기 위한 순서는 다음과 같다.

1. 탁상화면우의 [사용자등록부]를 두번 찰칵하여 파일열람기를 연다.
2. [보기]차림표의 [숨긴파일 표시]를 선택한다,
3. bash_profile을 편집기로 열고 다음의 행을 추가해놓는다.
   JAVA_HOME=/usr/java/jdk1.6.0
4. 체계를 재가입한다.
5. Eclipse를 재기동한다.




```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/usr/java/jdk1.6.0/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"
JAVA_HOME=/usr/java/jdk1.6.0

export USERNAME BASH_ENV PATH
```

그림 8-1. 환경변수 JAVA_HOME의 설정

8.4.2. 환경변수 Path

Java의 지령파일들이 들어있는 《 /usr/java/jdk1.6.0/bin 》등록부를 환경변수 PATH에 설정해놓는다. JAVA_HOME의 설정과 같이 하면 된다. 본래의 PATH설정행의 제일 마지막에 다음의 문자《:/usr/java/jdk1.6.0/bin》를 추가해놓는다(그림 8-2).

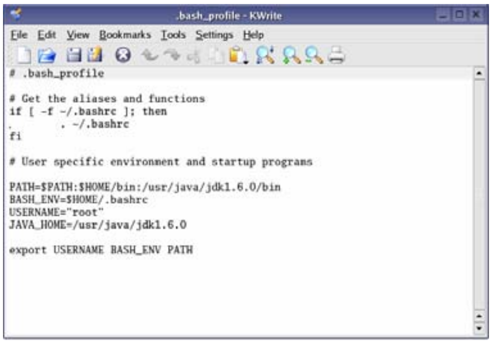


그림 8-2. 환경변수의 설정

제5절. Ant의 간단한 실행

Eclipse에서 Ant를 실행하는 방법은 여러가지가 있지만 가장 간단한 방법은 패키지열람기에서 build.xml을 오른쪽찰칵하여 현시되는 문맥차림표로부터 [Run]→[1 Ant Build]을 선택하는 방법이다. 이렇게 하면 선택된 건립파일의 기정목표가 즉시 실행된다. 표준출력이나 표준오유출력은 Console(조종탁)보기창에서 확인할수 있다(그림 8-3).



그림 8-3. Console보기창에서 결과의 확인

제6절. Ant의 기동구성

Eclipse에는 《기동구성》이라는 개념이 있다. 단순한 Java응용프로그람, JUnit시험클라스, 끼워넣기시험용의 실행시간작업대 등은 모두가 개개의 기동구성을 가지고있으며 실행할 때 필요한 여러가지 설정을 진행한다. 5절에서는 없었지만 Ant의 실행에도 [기동구성]이 존재하고 Ant실행의 고유설정을 할수 있다. 패키지열람기에서 build.xml을 오른쪽찰칵하고 [Run]→[2 Ant Build …]을 선택하면 [Ant기동구성]의 설정대화칸이 현시된다(그림 8-4).

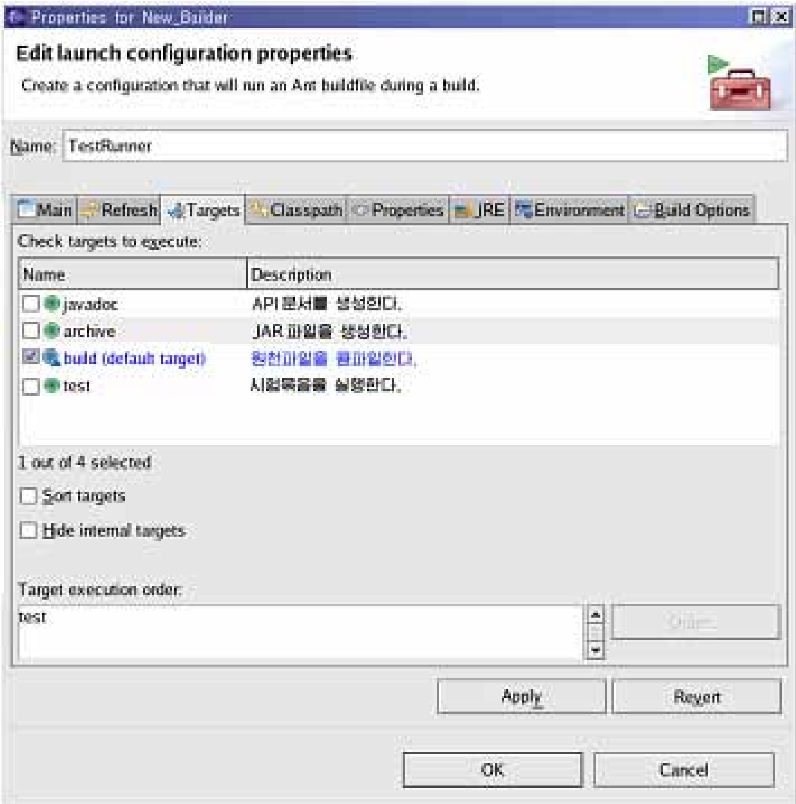


그림 8-4. [Ant기동구성]의 설정대화칸

이 대화칸에서 실행목표의 선택, 클라스경로나 속성 등 여러가지 설정을 진행할수 있다. 이 설정들은 개개의 기동구성으로서 보관된다. 전체 건립파일의 공통설정은 차림표떠로부터 [Window]→[Preferences]→[Ant]→[Runtime](아래에서는 **《Ant실행시간설정》**이라고 부른다.)에서 진행한다.

8.6.1. Main태브

[Ant기동구성]의 [Main]태브에서 건립파일의 장소, 지령행인수 등 기본적인 정보를 정의한다(그림 8-5).

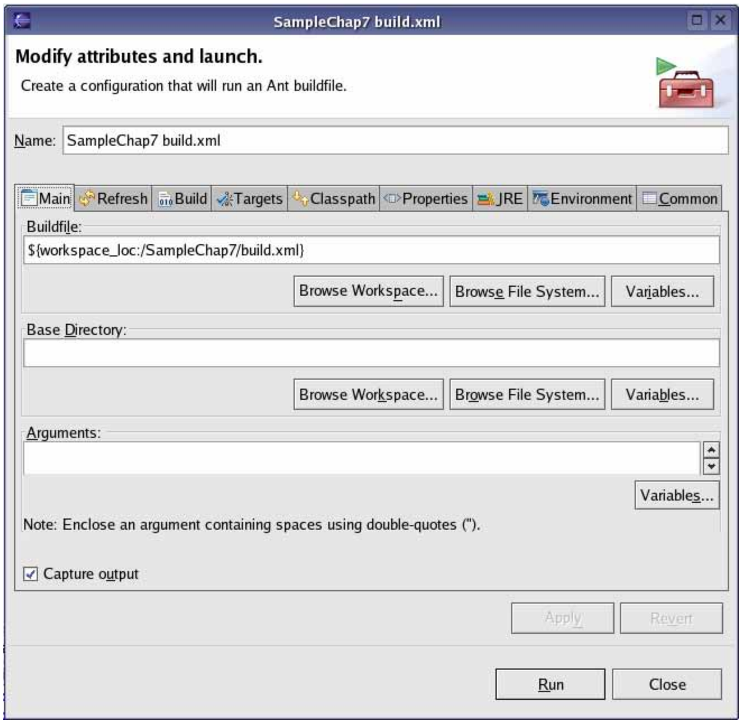


그림 8-5. Ant기동구성의 [Main]태브

[Build file(건립파일)]

건립파일의 위치이다. 이미 적당한 값으로 설정되여있기때문에 특별히 변경할 필요가 없다.

[Base Directory(기준등록부)]

건립파일안에서 상대적으로 지정하여야 할 경로이름들은 이 등록부를 기준으로 하여 정해진다. <project>요소의 basedir속성의 지정도 의미적으로는 같지만 이 방법이 우선시된다.

Ant의 속성 ${basedir}로서 건립파일안에서 참조할수 있다.

[Arguments(인수)]

Ant에 넘겨지는 지령행인수이다.

오른쪽의 [Variables]단추를 찰칵하여 프로젝트의 경로이름을 표시하는 변수를 삽입할수 있다.

8.6.2. Refresh태브

Ant의 실행에 있어서 작업공간상의 변화(파일의 작성, 지우기 등)가 있는 경우에 Eclipse가 인식하고있는 상태와 실제상태사이에 부정합이 발생한다. 구체적으로는 작성된 파일이 보기에 현시되지 않을수도 있고 제거된 파일이 보기에 남아있을수도 있다. 이것을 해소하기 위하여 수동으로 [Refresh]를 실행해도 좋지만 여기서 설정해놓으면 Eclipse가 자동적으로 진행한다(그림 8-6).

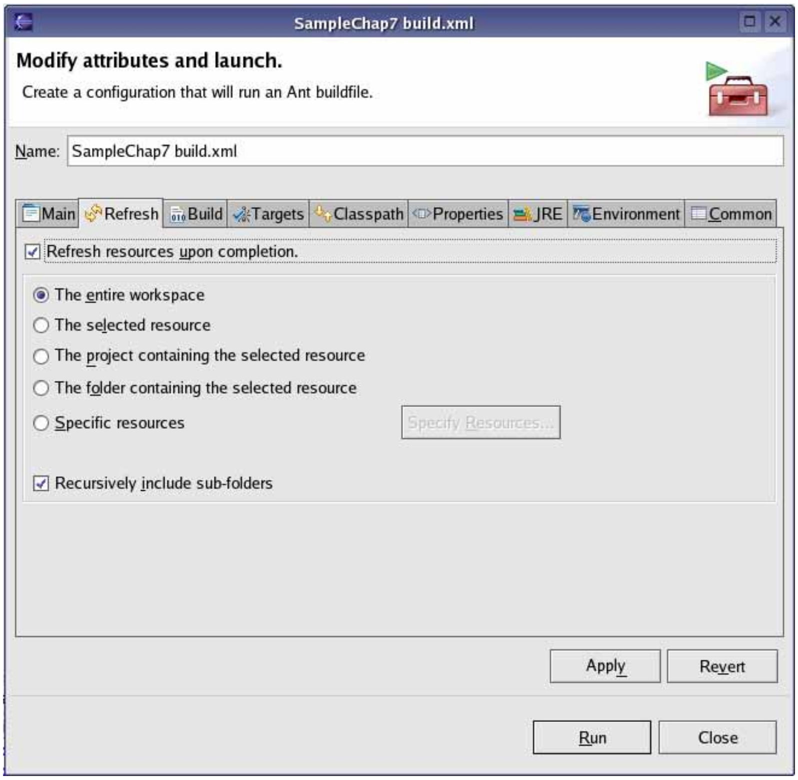


그림 8-6. Ant기동구성의 [Refresh]태브

· [Refresh resources upon completion(완료할 때의 자원의 갱신)]

이것을 선택하면 Ant의 실행을 완료할 때에 [Refresh]가 실행된다. 그때 갱신의 적용범위를 표현하는 선택단추들이 활성화된다. 기정은 [The entire workspace]로 되여있지만 작업공간상에 많은 프로젝트가 존재하는 경우에는 갱신에 상당한 시간이 걸리게 된다. 그런 경우는 [The project containing the selected resource(선택된 자원을 포함한 프로젝트)]를 선택하여 범위를 좁히는것이 좋다.

· [Recursively include sub-folders(재귀적으로 보조등록부까지 적용)]

이것을 선택하면 [Refresh]를 보조등록부까지 재귀적으로 적용한다. 기정으로서 선택되여있다.

8.6.3. Target태브

[Ant기동구성]대화칸에서 기정으로 선택되여있는 태브이다. 실행하려는 목표를 선택한다(그림 8-7).

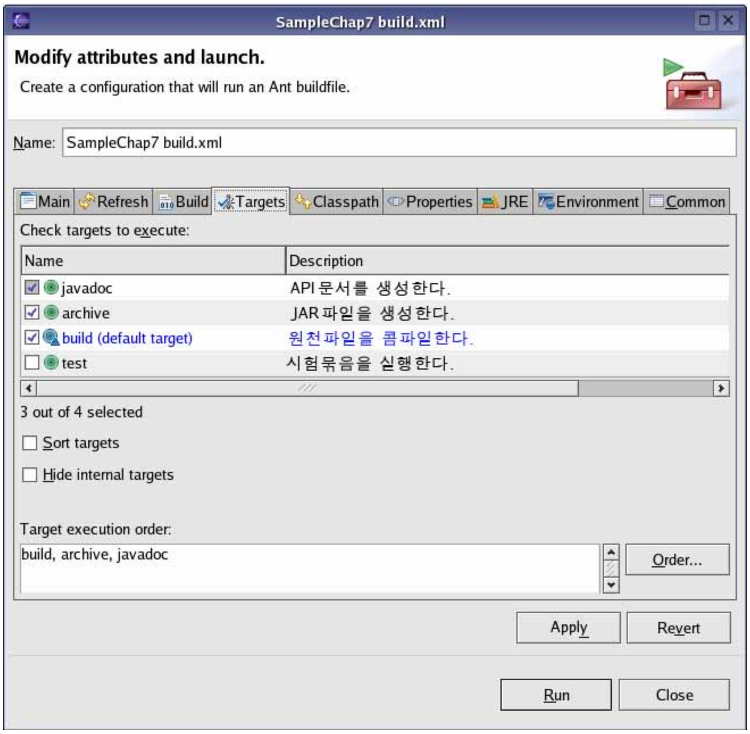


그림 8-7. Ant기동구성의 [Target]태브

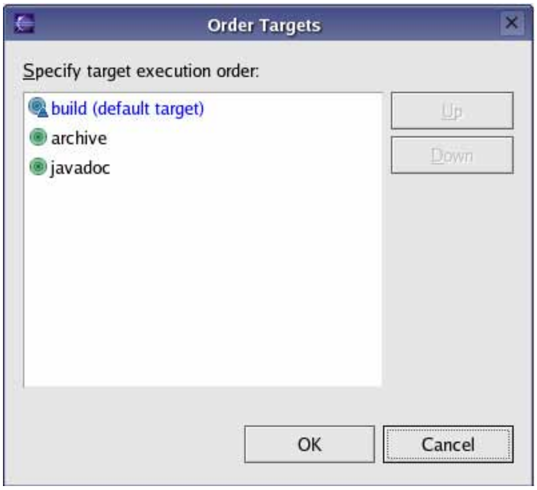


그림 8-8. 실행순서설정대화칸

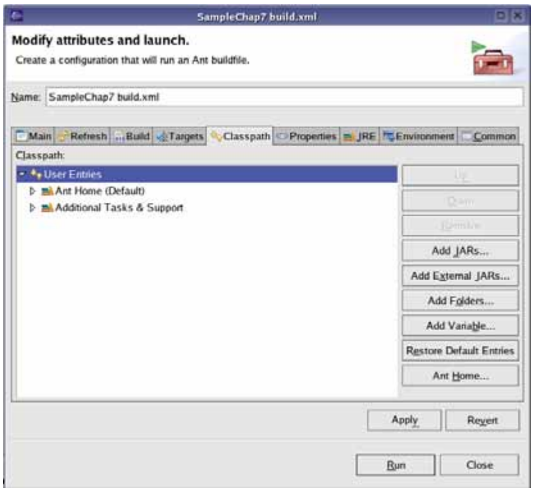


그림 8-9. Ant기동구성의 [Classpath]태브

· [Check targets to execute(실행목표선택)]

실행하려는 목표를 선택한다. 목표는 여러개 선택가능하다.

· [Target execution order(목표실행순서)]

2개이상의 목표를 선택한 경우는 [Order]를 찰칵하여 실행순서를 변경시킬수 있다(그림 8-8).

8.6.4. Classpath태브

Ant실행시 Java프로젝트안에서의 건립경로는 유효하지 않다. Ant실행용 클라스경로를 따로 정해주어야 한다. 전체 기동구성에서 유효로 하려는 경우에는 [Ant실행시간 설정]에서 정의한다. [Classpath]태브의 설정은 이 기동구성에서만 유효하다(그림 8-9).

이 기동구성에 대하여 클라스경로를 정의한다. [Add JARs]단추로 작업공간상의 프로젝트안에 있는 jar파일을 추가할수 있다. 또한 [Add External JARs]단추를 사용하면 파일체계상의 임의의 장소에 있는 jar파일도 추가할수 있다. [Up], [Down]의 단추로서 순서를 바꿀수 있다.

8.6.5. Properties태브

건립파일에서 참조된 속성을 정의한다. 초기상태에서는 [Use global properties as specified in the Ant runtime preferences(Ant실행시간설정에서 지정된 대역적속성을 사용)]이 선택되여 있는데 이것은 [Ant실행시간설정]에서 설정한 그대로이다. 선택를 없애면 이 기동구성의 고유속성을 정의할수 있다(그림 8-10).



그림 8-10. Ant기동구성의 [Properties]태브

· [Properties]

[Add property]단추로 속성의 이름과 값을 추가할수 있다. 이것은 <property>요소에서 name과 value속성을 지정하여 정의하는것과 같다. [Edit property]단추로 이미 존재하는 속성을 편집할수 있다.

· [Property files]

java.util.Properties형식의 속성파일을 등록할수 있다. 이것은 <property>요소에서 file속성을 정의하는것과 같다.

조선어(전각문자)를 포함한 파일은 native2ascii명령으로 유니코드로 변환해둘 필요가 있다.

8.6.6. JRE태브

Ant실행시에 사용하는 JRE에 관한 설정을 진행한다.

보통 이 설정을 변화시킬 필요가 없지만 다른 JRE를 사용할 경우에는 [Separate JRE]를 선택하여 다른 JRE로 절환할수 있다.

즉 선택가능한 JRE는 [Window]→[Preference]→[Java] →[Installed JRES]에 등록된 JRE들이다.(그림 8-11)



그림 8-11. Ant구성의 [JRE] 태브

8.6.7. Environment태브

앞에서 서술한 [JRE]태브에서 [Separate JRE]를 선택하여 Ant실행에 적용된 환경변수를 설정할수 있다.

[New…]를 찰칵하면 새로운 환경변수를 추가할수 있고 [Select…]를 찰칵하면 이미 존재하는 환경변수를 선택할수 있다. [Environment variables to set]란에 추가된 환경변수는 [Edit…]를 찰칵하여 값을 재편집할수 있다(그림 8-12).

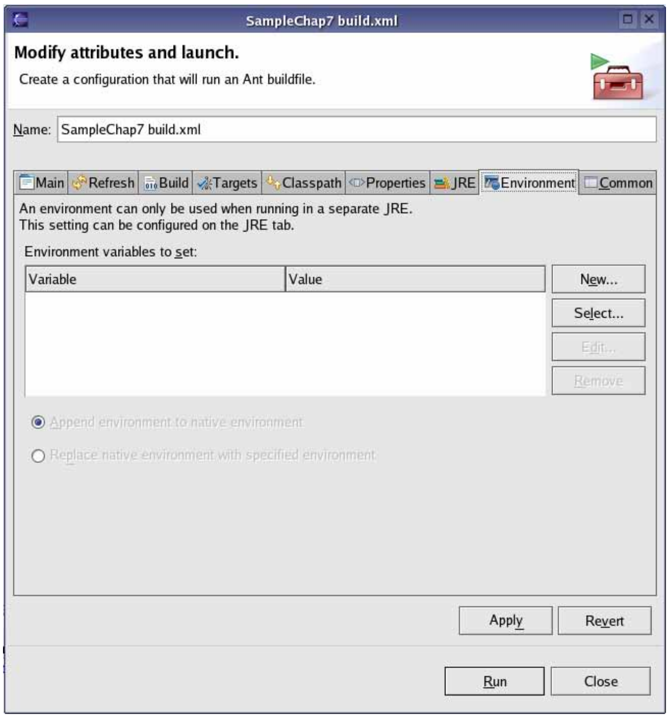


그림 8-12. Ant기동구성의 [Environment]태브

8.6.8. Common태브

기동구성의 실행 및 보존에 대한 설정을 진행한다.

・ [Type of launch configuration(기동구성의 형태)]

보통 [Local] 그대로 하여도 상관이 없다. [Shared]를 선택하면 기동구성을 XML 형식으로 작업공간상에 출력할수 있게 된다. 이 설정은 팀개발을 할 때 기동구성을 프로젝트에서 공유하는 경우에 편리하다.

・[Display in favorites menu(애호차림표에 표시)]

여기서 [External tools]를 선택하면 작업대의 [Run]→[External Tools]를 선택하여 현시되는 차림표에 이 기동구성이 추가되며 결과 간단히 실행할수 있게 된다.

・[Launch in background(배경에서 기동)]

이것을 선택하면(기정으로 선택되여있다.) Ant실행이 배경방식으로 진행된다. 특별한 리유없이 변경할 필요는 없다(그림 8-13).

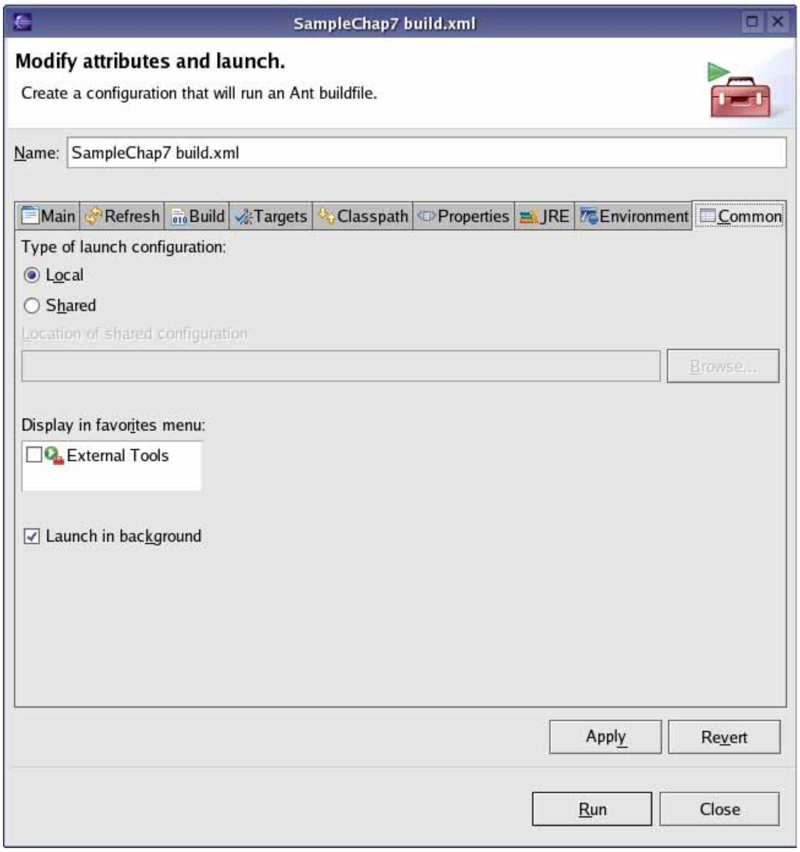


그림 8-13. Ant기동구성의 [Common]태브

제7절. [Ant]보기창

[Ant]**보기창**은 여러개의 건립파일을 일괄적으로 관리하여 여러가지의 목표를 간단히 실행할수 있는 편리한 창이다.

차림표띠로부터 [Window]→[Show View]→[Ant]를 선택하면 [Ant]보기창이 현시된다. 초기상태에서는 내용이 하나도 없는데 이것은 건립파일이 한개도 등록되여있지 않았기때문이다(그림 8-14).

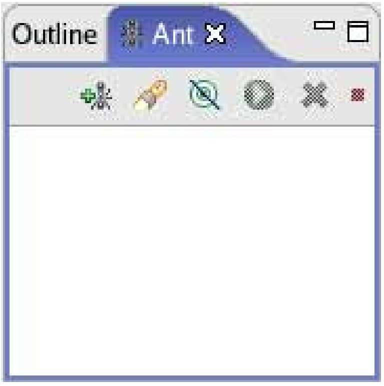


그림 8-14. [Ant]보기

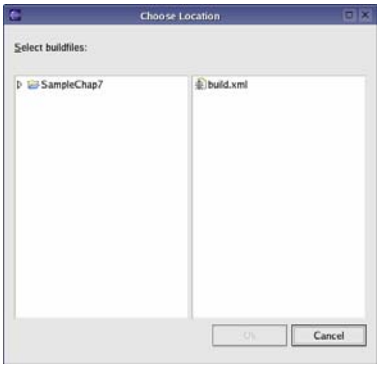


그림 8-15. 건립파일의 선택화면

여기에 건립파일을 등록한다. 그림 8-14의 [Add Buildfiles]단추를 찰칵한다(그림 8-15). 보기창에 추가하려는 건립파일을 선택한다. 그다음 [OK]를 찰칵하면 [Ant]보기창에 넣어진 건립파일(프로젝트)이 현시된다(그림 8-16). 이 상태에서 그림 8-16의 [Run the Default Target]단추를 찰칵하면 이 프로젝트의 기정목표인 build가 실행된다. 또한 ▷단추를 찰칵하여 프로젝트를 전개하고 다른 목표를 실행할수도 있다. 실례로 API문서를 생성하는 javadoc목표를 실행해본다. [Run the Default Target]단추를 찰칵하면 [Console]에 Ant로부터 출력된 문자렬내용이 현시된다(그림 8-17).



그림 8-16. 건립파일을 추가한 Ant] 보기

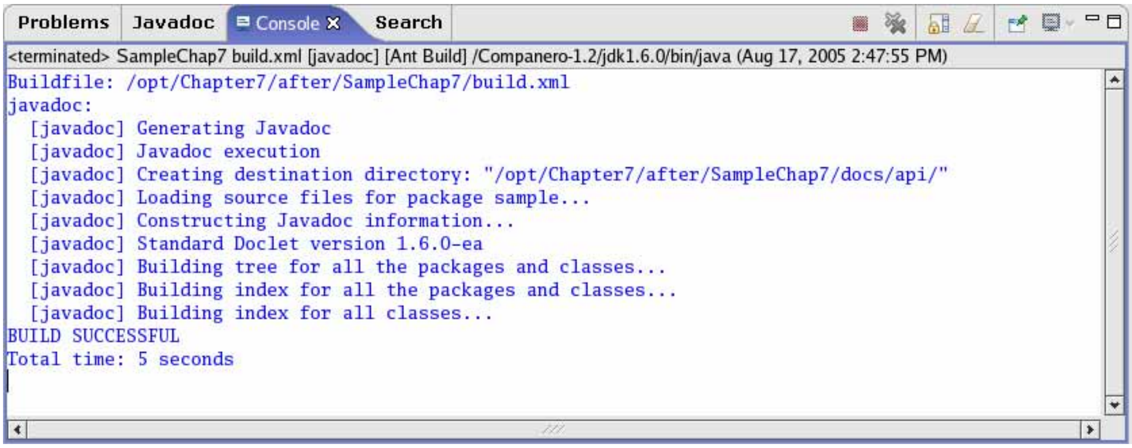


그림 8-17. 실행결과

정확히 실행되는가를 Web열람기에서 확인해본다(그림 8-18).

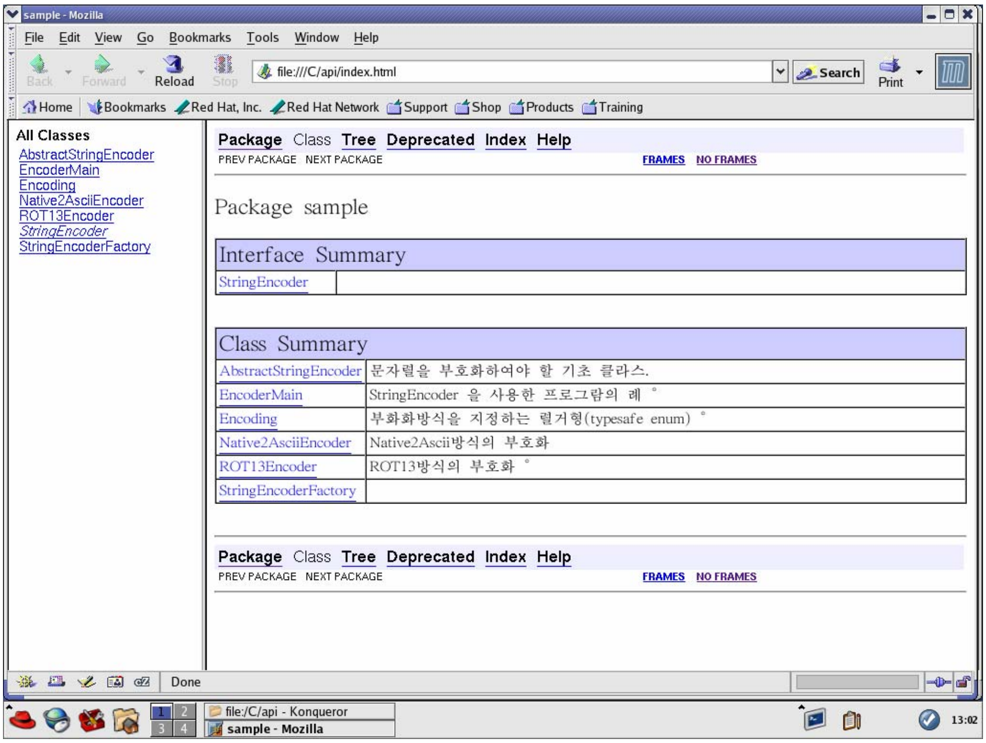


그림 8-18. 생성된 API문서

제8절. Ant편집기

지금까지 Ant의 실행과 기동구성의 상세한 설정을 하였는데 이제부터는 건립파일에 대한 편집을 해보기로 한다.

Eclipse에는 《 Ant편집기 》라고 하는 건립파일작성용편집기가 갖추어져있다. build.xml을 찰칵하여 Ant편집기에 이 파일을 연다(그림 8-19).

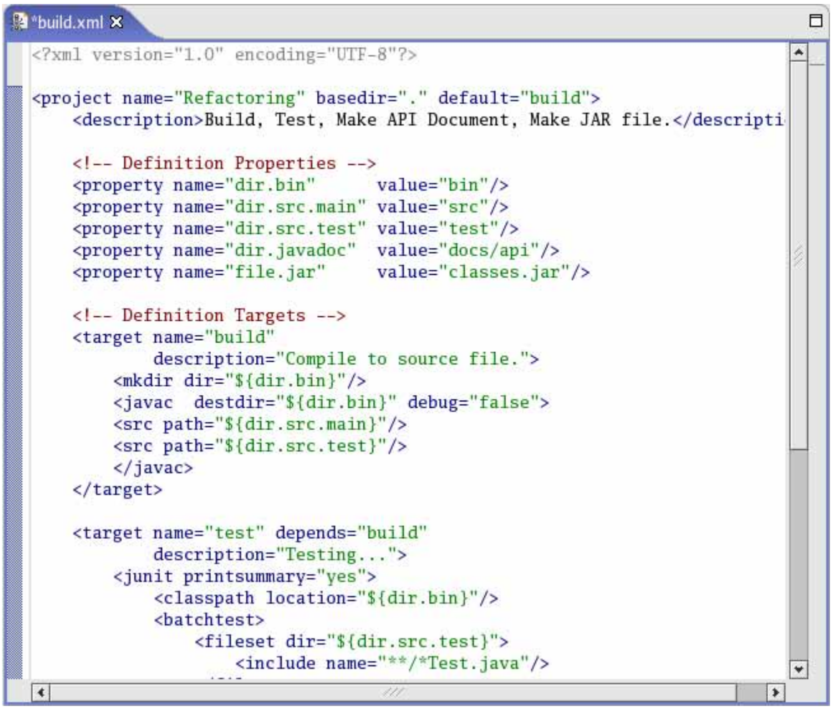


그림 8-19. Ant편집기에서 build.xml을 열었을 때

요소나 속성, 지령 등이 밝은 색으로 표시됨으로 보기가 쉽다.

그림 8-19의 build.xml에 목표를 한개 추가해본다.

build나 javadoc, archive 목표들의 실행과정에 생성된 파일을 모두 지워서 프로젝트를 깨끗한 상태로 만드는 《clean》목표를 하나 만들기로 한다. 편집기의 유표를 archive목표의 마지막까지 가지고가서 새로운 목표를 정의한다. 내용편집조력기능을 사용하여 최대한 빨리 해본다. 《<tar>》까지 입력하고 Ctrl+Space건을 누르면 <target>가 나타나며 그에 따르는 설명도 표시된다. [Enter]건을 누르면 마지막 태그까지 일식으로 입력되고 속성의 입력을 하기 위하여 시작태그와 마지막태그의 가운데로 유표가 이동한다.

[public tartget]를 선택한다(그림 8-20).

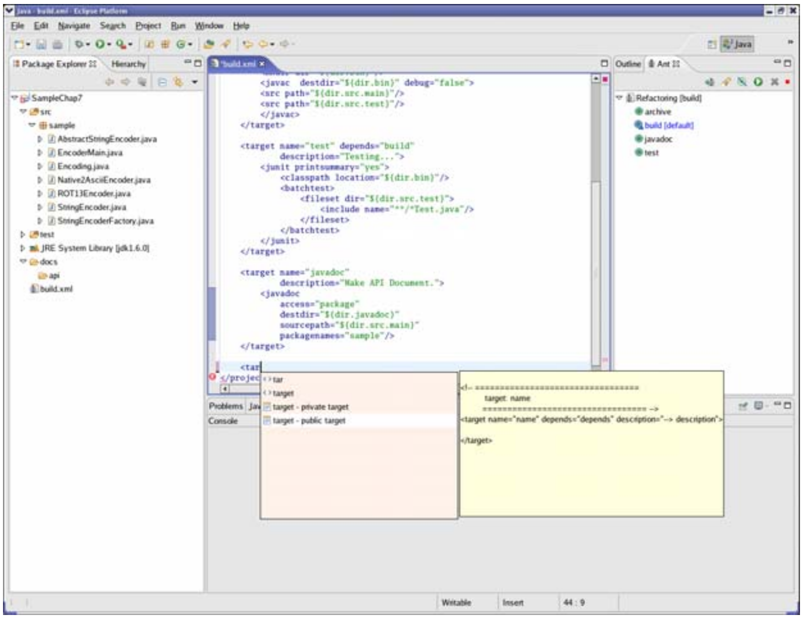


그림 8-20. 내용물방조

이것으로 쉽게 target태그의 골격이 완성되였다.
유표는 자동적으로 설명표식란의 [target:]행의 마지막으로 이동한다(그림 8-21).

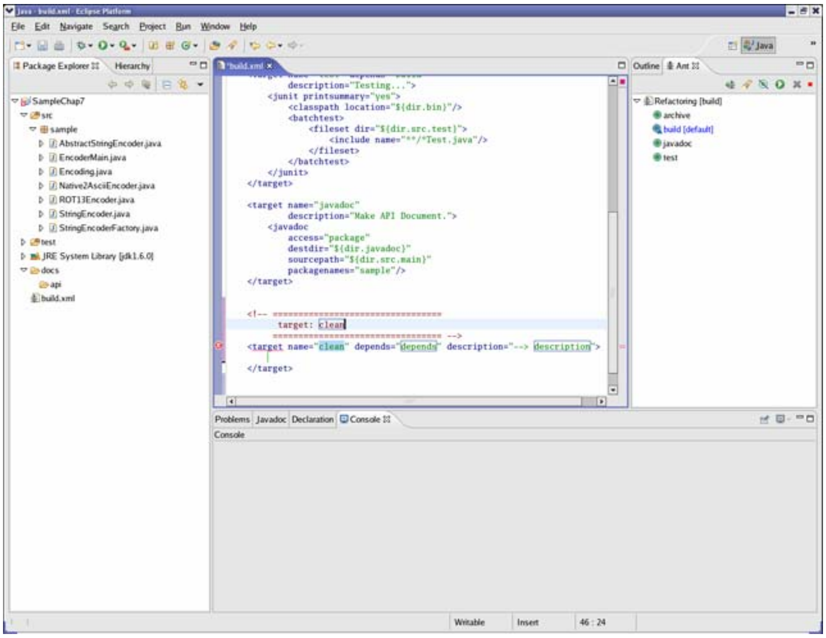


그림 8-21. 내용편집조력기능을 리용한 결과

depends속성도 자동적으로 생성되였지만 이 목표에서는 불필요한 속성이므로 제거한다. descriptions속성에는《건립에 의해 생성된 등록부를 제거한다.》라는 설명문을 써놓는다. 이것으로 목표의 틀거리가 완성되였다.

다음은 실제의 과제들을 서술하면 된다.

이 목표에서 다음의 3개를 제거하여야 한다.

- build에서 생성된 .class파일들

${dir.bin}에서 정의된 출력등록부자체를 지우기

- javadoc에서 생성된 문서

${dir.javadoc}에서 정의된 문서등록부자체를 지우기

- archive에서 생성된 jar파일

${file.jar}에서 정의된 jar파일을 지우기

실제로는 Ant의 과제 <delete>를 리용한다.

이번에는 속성편집방조기능으로 해본다.

《<delete dir= "${dir》까지 입력하고 Ctrl+Space건을 누른다. 그러면 그림 8-22와 같은 속성방조기가 현시된다.

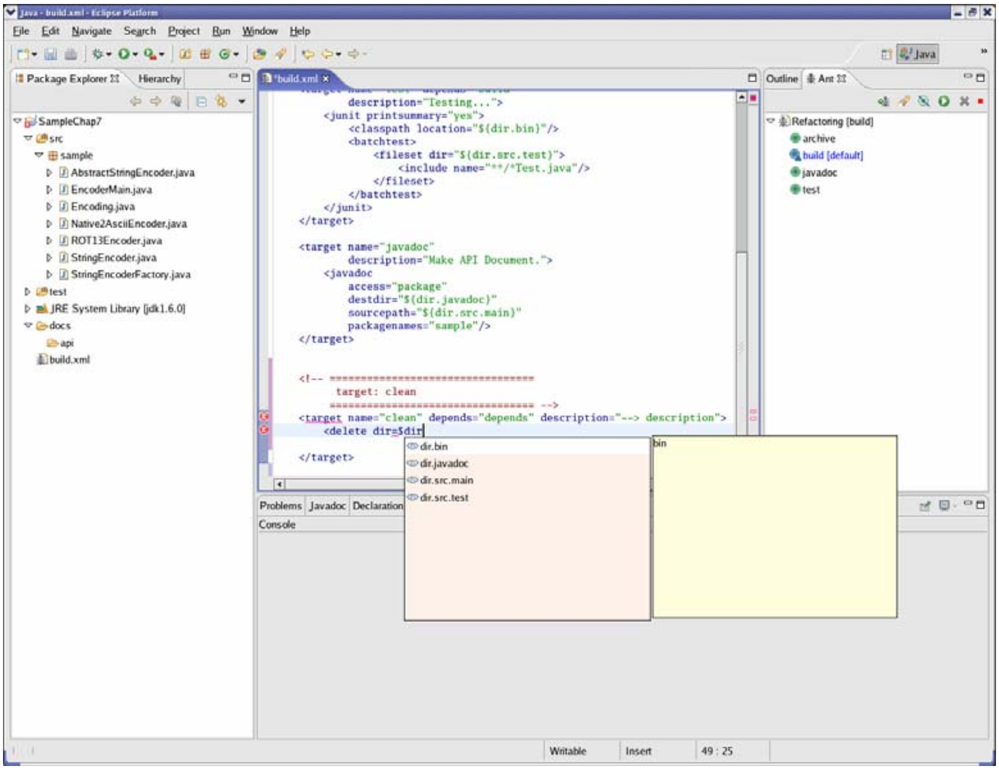


그림 8-22. 속성방조기

우선 .class파일출력등록부 $(dir.bin)을 선택해본다. 꼭같이 ${dir.javadoc}, ${file.jar}도 지워버리도록 과제에 입력하면 clean과제가 완성된다.

```
<!-- ===================================
   target : clean
======================================= -->
<target name= “clean” description= “건립에 의해 생성되는 등록부를 지운다.” >
   <delete dir= “${dir.bin}” />
   <delete dir= “${dir.javadoc}” />
   <delete file= “${file.jar}” />
</target>
```

건립파일에 추가된 clean목표를 시험해본다.

변경한 build.xml파일을 보존하고 [Ant]보기창에서 《clean》을 선택하여 실행한다(그림 8-23).

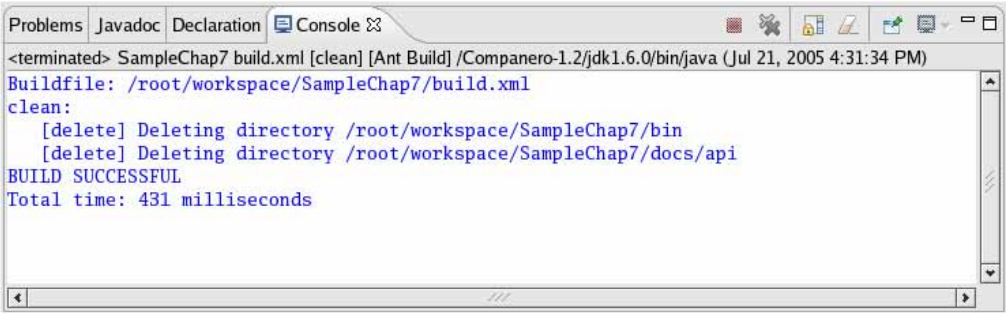


그림 8-23. clean목표의 실행

이처럼 Ant편집기를 사용하면 건립파일을 빠르고도 정확히 만들수 있다. 또한 매개 과제의 기능설명, 속성의 종류, 기정값 등은 Ant의 안내서를 보지 않아도 알수 있으며 속성이름의 타자입력실수와 같은 오유를 범하지 않을수 있다.

제9절. Ant로 완성된 파일건립

그러면 프로젝트에 대하여 build목표를 콤파일해본다.



그림 8-24. 콤파일실패

Eclipse상에서 문제가 제기되여 원천코드의 콤파일이 실패하였다. 이것은 [Classpath]태브에서 설명한 클라스경로설정에 문제가 있기때문이다. JUnit의 JAR파일이 없으므로 JUnit관련클라스를 참조하고있는 부분이 모두 오유로 되여버렸다. 이것을 해결하기 위해서는 Ant의 실행클라스에 JUnit의 jar파일을 추가하여야 한다.

파일추가방법에는 2가지가 있다.

[Ant실행시간설정]에서 추가한다.

[8.6.4 Classpath태브]에서 설명한것과 같이 건립파일의 기동구성에서 추가한다.

기동구성에 대하여서는 이미 설명하였기때문에 이번에는 [Ant실행시간 설정]에서의 파일추가방법에 대하여 보기로 한다.

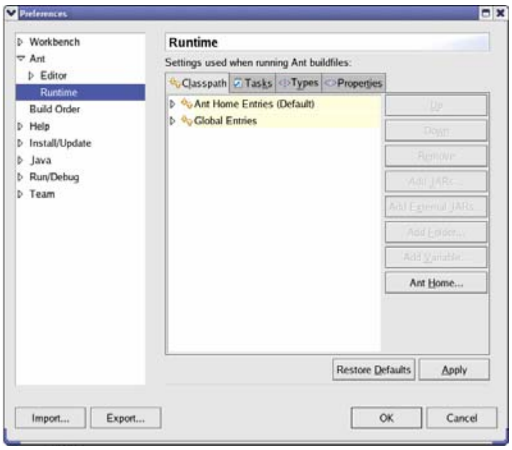


그림 8-25. Ant실행시간설정

차림표띠로부터 [Window]→[Preferences]를 선택하여 현시되는 대화칸에서 왼쪽의 [Ant]→[Runtime]이라는 차림표항목을 선택한다(그림 8-25).

[Global Entries]을 선택한 다음 [Add External JARs]단추를 찰칵하여 《/kut/eclipse/plugins/org.junit_3.8/junit.jar》를 추가한다. 다음 [OK]단추를 찰칵하여 설정화면을 열고 다시 한번 build목표를 실행시켜 본다(그림 8-26).

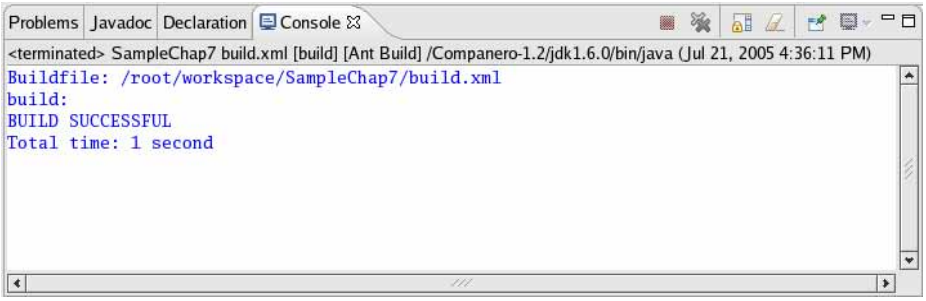


그림 8-26. Build목표의 실행

제10절. Ant와 Junit

마지막으로 JUnit의 시험클라스를 실행하는 test목표를 시험해본다.

이 목표의 실행을 Java프로젝트의 건립프로쎄스의 한부분으로서 조합시켜 완전작성을 진행할 때마다 JUnit로 시험을 진행하는 기능을 실현해본다.

8.10.1. test목표

목록 8-1의 build.xml로부터 test목표만 뽑아서 아래와 같이 표시한다. 먼저 이 목표의 구성을 해석해본다.

```
<target name= “test” depends= “build”
description= “시험을 진행한다.” >
<junit printsummary= “yes” >
   <classpath location= “${dir.bin}” />
   <batchtest>
          <filetest dir= “${dir.src.test}” >
                 <include name= “**/*Test.java” />
          </filetest>
   </batchtest>
</junit>
</target>
```

JUnit클라스에 대해서 printsummary= “yes” 는 test결과의 요점을 간단한 쓴 설명문을 표시하려고할 때 지정한다. 여기까지는 단순하지만 그 다음에 보면 약간 추가된 부분이 있다.

우선 <classpath>요소가 있다. 이것은 JUnit로부터 시험대상클라스가 보이도록 하기 위한것이다. <batchtest>요소는 여러가지 시험클라스를 한번에 실행하도록 하기 위한것이며 실제로 시험클라스들을 지정해주고있는것은 바로아래의 <fileset>요소이다. <fileset>는 dir속성에서 지정된 등록부를 기초로 하여 <include>요소의 name속성에서 지정된 문자렬과 결합시켜 파일묶음을 나타낸다. 《**》기호로 전체 보조등록부를 재귀적으로 검색한다.

종합해보면 이 test목표는 dir속성에서 지정된 등록부아래의 보조등록부를 재귀적으

로 검색하고 이름이 Test로 끝나는 클라스를 모두 검출하여 시험을 진행한다. test목표를 실행해보면 결과는 그림 8-27과 같다.

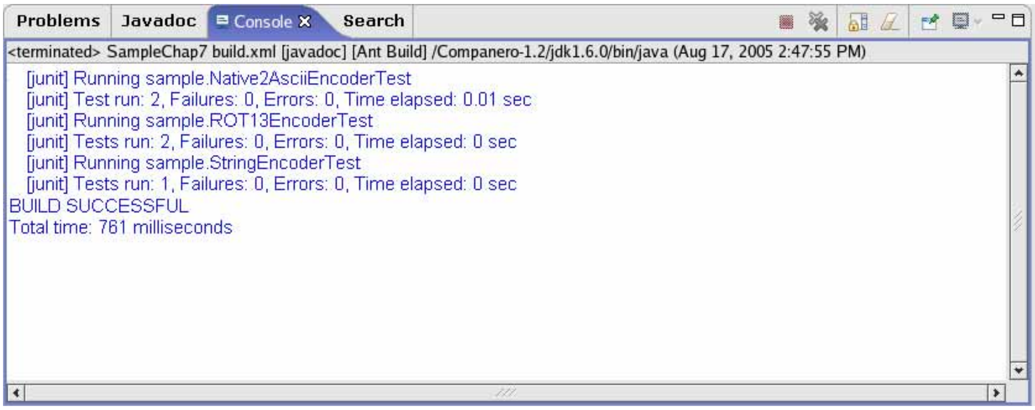


그림 8-27. Test목표의 실행

8.10.2. 시험의 자동화

코드를 작성한 다음 시험하고 다시 수정한 다음 시험을 또 진행한다. 이러한 경우 콤파일할 때마다 자동적으로 시험이 진행되도록 하려면 어떻게 해야 하는가에 대하여 보기로 한다. 이 문제는 Eclipse와 Ant를 서로 결합하여 간단히 실현할수 있다. Eclipse에는 외부도구건립이라는 기능이 있어 프로젝트의 작성작업을 통속화할수 있다. 이것을 리용하여 작성과 동시에 Ant목표의 실행을 진행할수 있다.

먼저 [SampleChap7]프로젝트를 오른쪽찰칵하여 [property]를 선택한다. 현시된 대화칸의 왼쪽에서 [Builder]를 선택하고 [New]을 찰칵한다. 이 대화칸에서 새로운 건설도구를 추가할수 있다(그림 8-28).

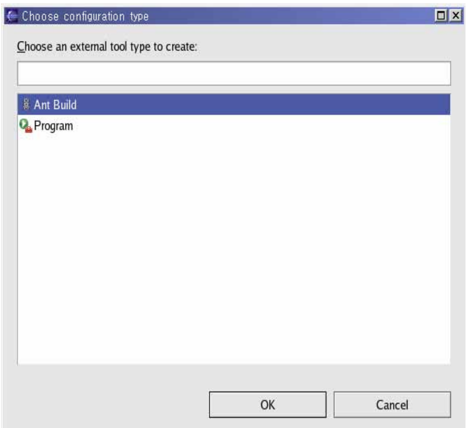


그림 8-28. 외부도구건립대화칸

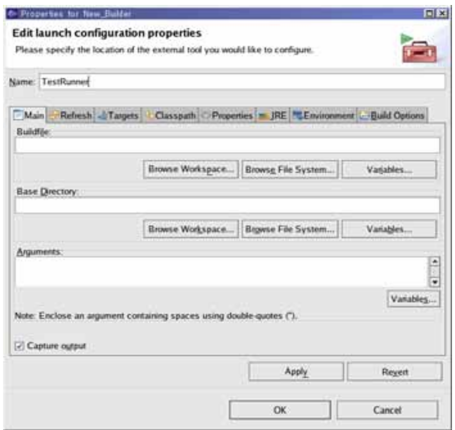


그림 8-29. 외부도구건립 기동구성

여기서 [Ant Build]를 선택하고 [OK]단추를 찰칵하면 [Edit launch configuration properties]대화칸이 나타난다. 먼저 건립도구이름을 결정한다. 이름은 《TestRunner》라고 입력하고 [Capture output]를 잊지 말고 선택하여야 한다. 그렇게 하지 않으면 실행결과를 확인할수 없다(그림 8-29). [Buildfile]에 build.xml를 설정한다. 이 파일을 설정하기 위해서는 [Browser Workspace]단추를 찰칵하여 build.xml을 선택하여야 한다. [OK]단추를 찰칵한다(그림 8-30).

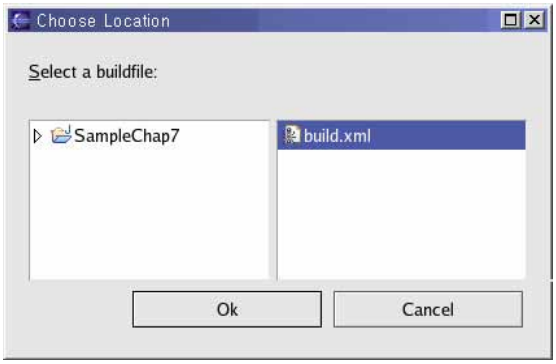


그림 8-30. 건립파일의 선택

다음으로 [Targets]태브로 절환시켜 실행하려는 목표를 설정한다. 기정목표로 build가 선택되여있는데 이것을 해제하고 test를 선택한다(그림 8-31).

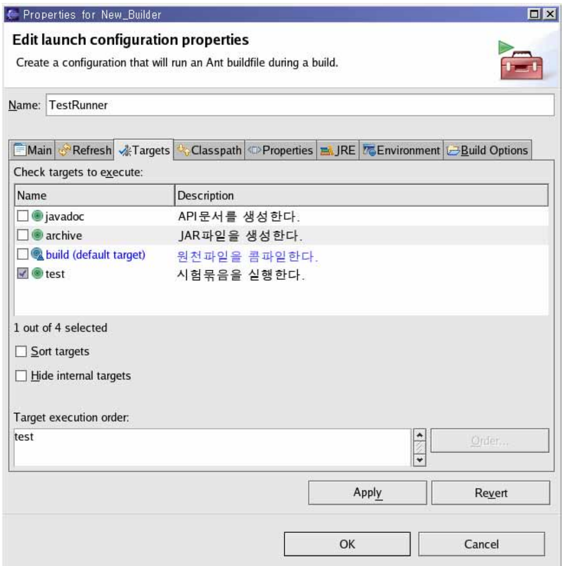
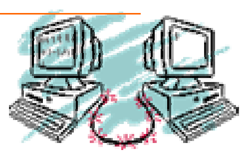


그림 8-31. build목표를 해제하고 test목표를 선택



마지막으로 [Build Options]태브를 선택한다. 여기서는 새롭게 추가하는 건립도구를 어떤 시점에서 실행시키겠는가를 설정한다. 표준으로는 [After a "Clean"]와 [During Manual Builds]가 선택되여있다. [During Auto Builds]는 특별한 리유가 없는 한 변경시키지 않는것이 좋다. 또한 [Launch in background]을 선택해놓으면 Ant가 배경에서 실행되게 된다(그림 8-32).

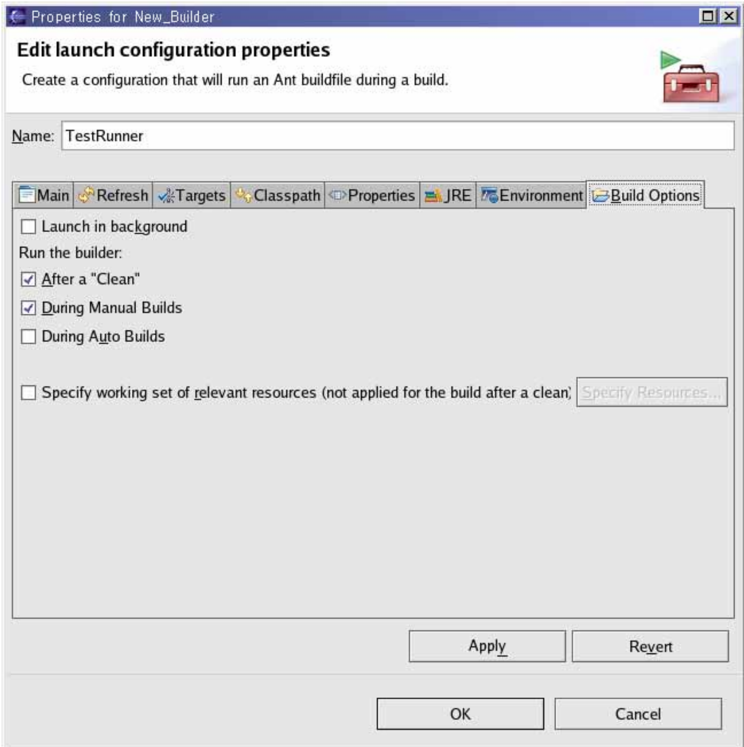


그림 8-32. Build Option설정

이상으로 설정을 끝마친다. [OK]단추를 찰칵하여 즉시 시험해본다.

차림표띠로부터 [Project]→[Clean…]를 선택한다. 여기서는 [Clean all projects]가 선택되여있다. [OK]단추를 찰칵하면 지우는 동작이 실행된다(그림 8-33).

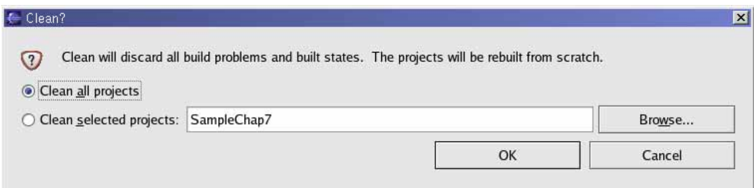


그림 8-33. Clean의 실행

이 방법을 리용하면 응용프로그람건립과 꼭같이 API문서를 갱신하기도 하고 또 건립하기 전에 불필요한 파일을 지우기도 하는 등 여러가지 처리를 할수 있다.

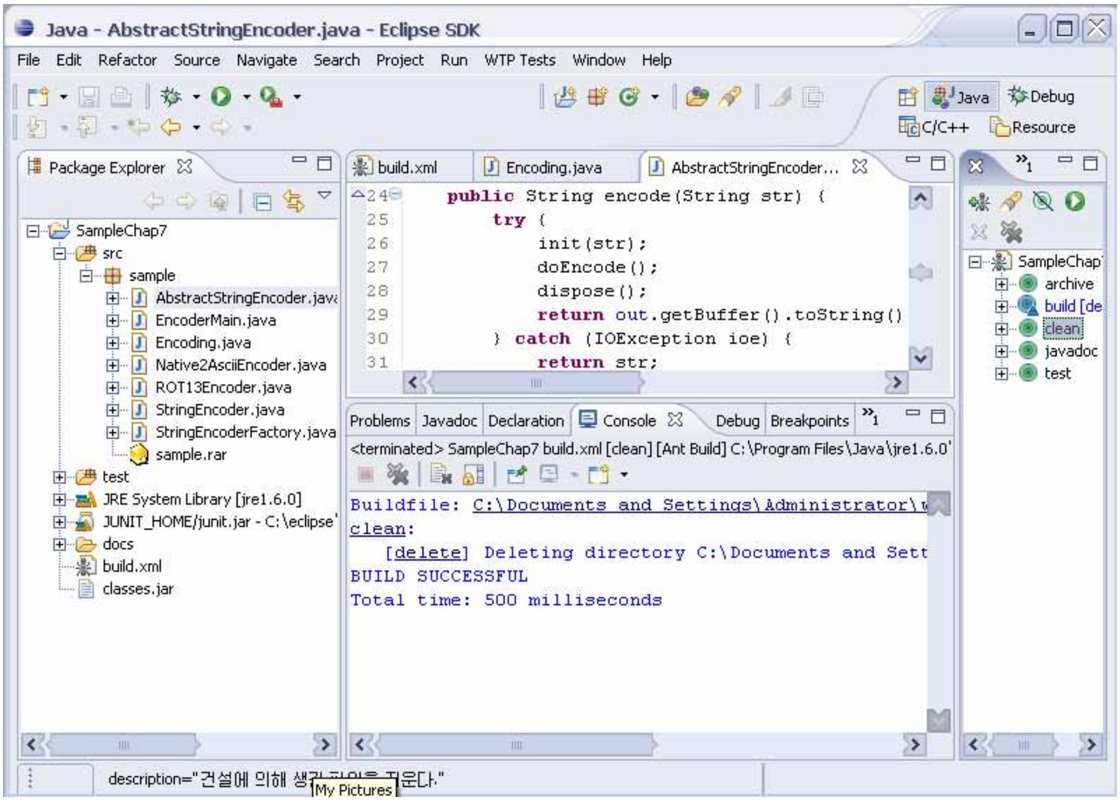


그림 8-34. Clean의 성공

제9장. 공동판본체계에 의한 팀개발

제1절. 팀개발

9.1.1. 공동판본체계(CVS)란 ?

CVS(Concurrent Version System)란 여러명의 개발자가 참가하는 응용프로그람개발(팀개발환경 혹은 팀프로그람작성이라고 함)에 있어서 응용프로그람코드의 판본관리를 진행하기 위한 쏘프트웨어이다. Eclipse에는 이 CVS를 리용하기 위한 기능이 제공되여있다.

9.1.2. 역할위주(role base)의 개발법

최근 대규모 웨브응용프로그람 또는 J2EE응용프로그람개발에서는 역할위주(role base)의 개발이 주류를 이루고있다.

역할위주의 개발법이란 응용프로그람개발에 참가하는 매 개발자의 개발분야나 임무를 명백히 하고 여러개의 프로그람구성요소의 개발을 동시병행적으로 진행하는 형태의 개발방법을 말한다. 프로젝트의 규모도 수명정도가 아니라 수십～수백명단위로 포함되는 경우가 많고 그 사람들로 팀을 무어 개발을 진행한다. 그 중에서도 개발되는 응용프로그람(HTML파일, JSP파일, JavaBeans프로그람, Enterprise JavaBeans(EJB)프로그람 등)의 구성요소는 대단히 많고 필요한 숙련도 다양하므로 이것들을 한 사람 또는 여러명으로 개발하는것은 힘들다.

실례로 사용자가 보는 화면의 설계를 맡은 사람(페지설계가라고 부른다.)은 화면의 설계 및 개발만 하고 그 화면을 사용하기 위한 프로그람구성요소에는 일체 관계하지 않는다. 이러한 개발방법은 응용프로그람개발에 관해서는 매 사람이 자기의 전문분야에만 힘을 집중하도록 할수 있지만 최종적으로는 하나의 응용프로그람으로서 동작하도록 하여야 하기때문에 어떤 의미에서는 그 구성요소들이 개발자전체성원에 대하여 공통으로 필요한 구성요소로 된다. 이것은 전체 개발성원이 적절한 시기 필요한 프로그람구성요소에 접근조종할수 있도록 관리를 진행할 필요가 있다는것을 의미한다. 또한 개발작업을 진행하는 과정에 여러 사람들의 요구에 의하여 프로그람코드에 변경이 가해지기때문에 이 변경점들을 적절히 관리하는것도 중요하다.

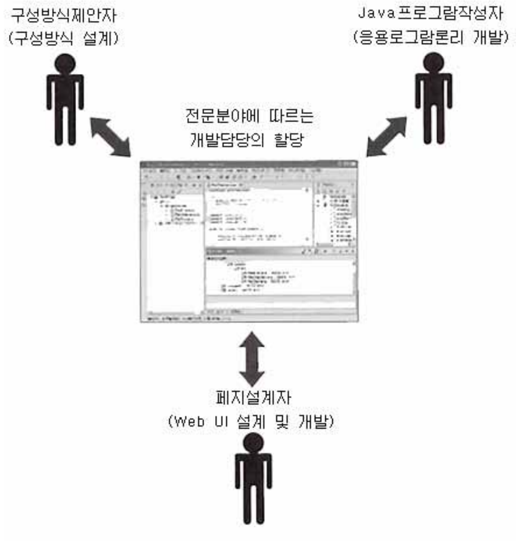


그림 9-1. 역할위주의 개발

CVS에서는 이와 같은 개발환경을 지원하며 자원보관(Repository) 및 판본추가기능을 제공해준다. 자원보관은 일종의 자료기지를 의미하며 개발작업에서 작성된 프로그람코드가 도중에 보관된다. 그때에 프로그람코드의 판본을 붙이는 방법으로 판본관리를 진행한다. CVS는 말단기/봉사기형식으로 동작하며 자원보관은 CVS에 의해 관리된다.

다음절에서 CVS의 도입방법에 대하여 설명한다.

Eclipse는 CVS봉사기에 접근조종하고 보관고안에 보관된 자료에 접근조종할수 있는 말단기기능을 제공하고있다. Eclipse를 사용하여 CVS봉사기에 접근하는것으로써 팀개발환경을 마련할수 있다.

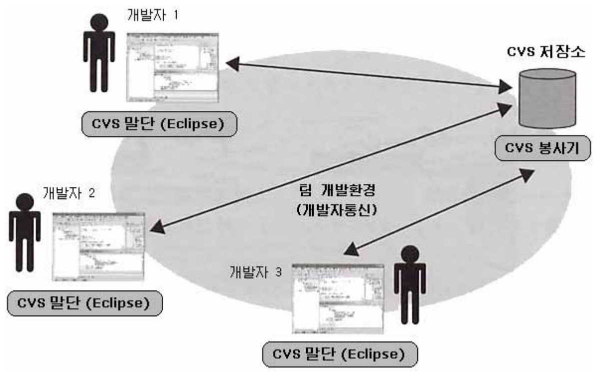


그림 9-2. CVS리용환경

9.1.3. CVS를 리용한 개발방법

CVS를 리용한 팀개발방법은 다음과 같다.

① CVS봉사기 및 보관고를 준비한다.

이미 서술한바와 같이 CVS는 말단기/봉사기로서 동작하기때문에 우선 봉사기를 준비하여야 한다.(봉사기를 준비하는 과정에 보관고가 만들어진다.) 이것에 대해서는 후에 자세히 설명한다.

② 개발코드의 기초(초기판본)를 작성한다.

개발의 기초코드를 작성한다. 이 코드는 전체 개발작업의 기초로 되고 이것에 대한 기능의 추가/갱신을 하면서 작업을 진행하는것으로 된다.

③ 기초코드를 CVS의 보관고안에 보관한다.

작성한 기초코드를 CVS의 보관고안에 보관한다. 이것에 의해 CVS보관고안에 프로그람코드가 작성된다.

④ 필요에 따라 CVS보관고로부터 필요한 코드를 얻고 추가/갱신을 진행한다.

CVS보관고로부터 코드를 얻고 필요한 추가/갱신을 진행한다.

보관고로부터 얻어낸 코드는 작업자의 국부적인 환경에 복사된다. 따라서 코드에 대한 추가/갱신의 수정작업은 다음에 CVS보관고에 보관할 때까지는 보관고에 반영되지 않는다.

⑤ 작업을 완료하면서 다시 한번 보관고로 수정한 코드를 되돌린다.

국부공간에서의 작업을 끝내면서 그것을 다시한번 보관고에 보관한다.

보통 개발작업이 완전히 끝날 때까지는 ④, ⑤단계가 반복되여 진행된다.

제2절. CVS봉사기의 설치

CVS를 리용한 팀개발환경을 구축하자면 우선《CVS봉사기》를 설치하여야 한다. 일반적으로 CVS봉사기는 UNIX계의 조작체계상에서 작성되는 경우가 많지만 여기서는 Windows환경에서 개발작업을 할수 있도록 cvsnt라는 무료쏘프트웨어를 사용하여 Windows XP상에 봉사기를 설치한다.

9.2.1. cvsnt의 판본

cvsnt의 최신 안정판본은 2.0.58a이지만 이 판본은 Eclipse와 조합시켜놓으면 일부 불합리한 점이 있으므로 여기서는 다른 안정판인 2.0.51d를 사용한다.

9.2.2. CVS봉사기의 설치

현재 CVS봉사기쏘프트웨어는 Windows용이므로 설치 및 경로는 Windows XP경우에 따른다. cvsnt-2.0.51d.exe를 실행한다(그림 9-3).



그림 9-3. 설치프로그람의 첫 화면

[Next]단추를 찰칵하면 GNU GENERAL PUBLIC LICENSE의 동의확인대화칸이 표시된다. 동의하는 경우에는 [I accept the agreement]를 선택한 다음 [Next]단추를 찰칵한다. 다음 디스크상의 어느 등록부에 cvsnt를 설치하겠는가를 설정한다. 여기서는《C:\Program Files\cvsnt》등록부에 설치한다(그림 9-4).

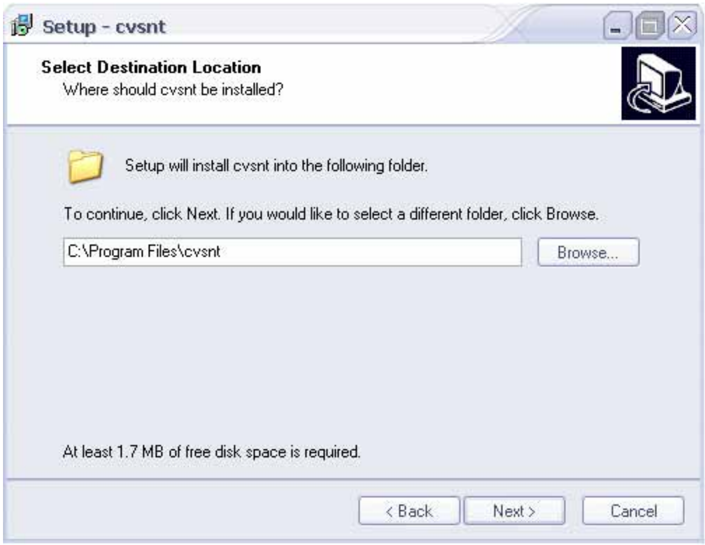


그림 9-4. 설치장소의 설정

설치하려는 구성요소를 선택한다(그림 9-5).

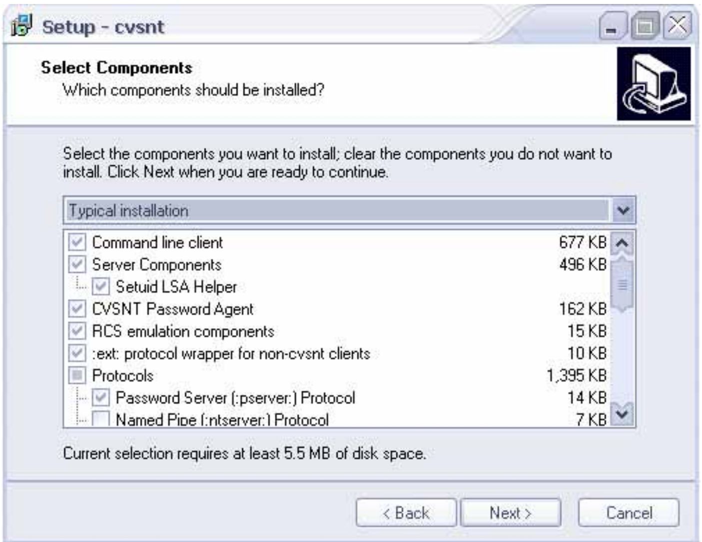


그림 9-5. 설치요소들의 선택

[Next]단추를 찰칵하고 설치하려는 등록부이름을 설정한다. 이 값도 기정값으로 설정한다. [Next]단추를 찰칵하고 제공하는 봉사 등을 설정한다. 이것도 기정값으로 해도 좋다. 마지막으로 확인화면이 나타나는데 여기서 [Install]를 찰칵하면 설치가 시작된다(그림 9-6).

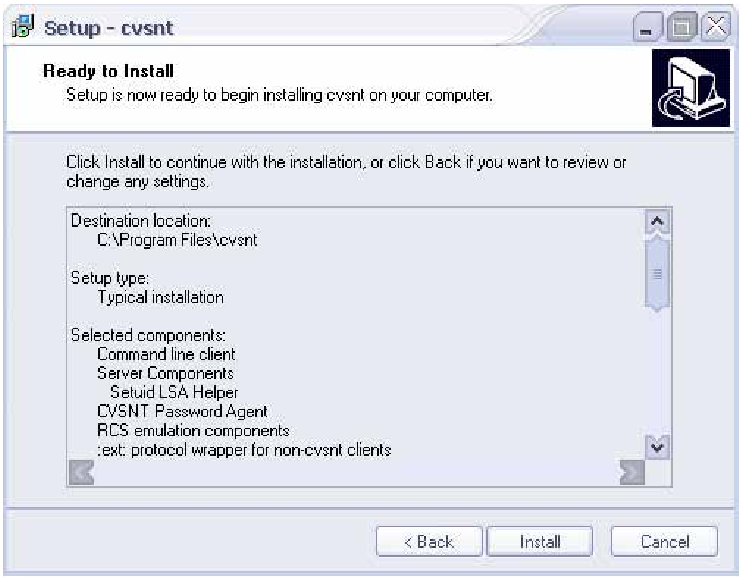


그림 9-6. 설치작업개시전 확인대화칸

이렇게 하면 cvsnt가 설치된다. 지정한 장소에 실행파일이 설치되였을 뿐아니라 그림 9-7에서 보여준것처럼 2개의 봉사가 등록된다.

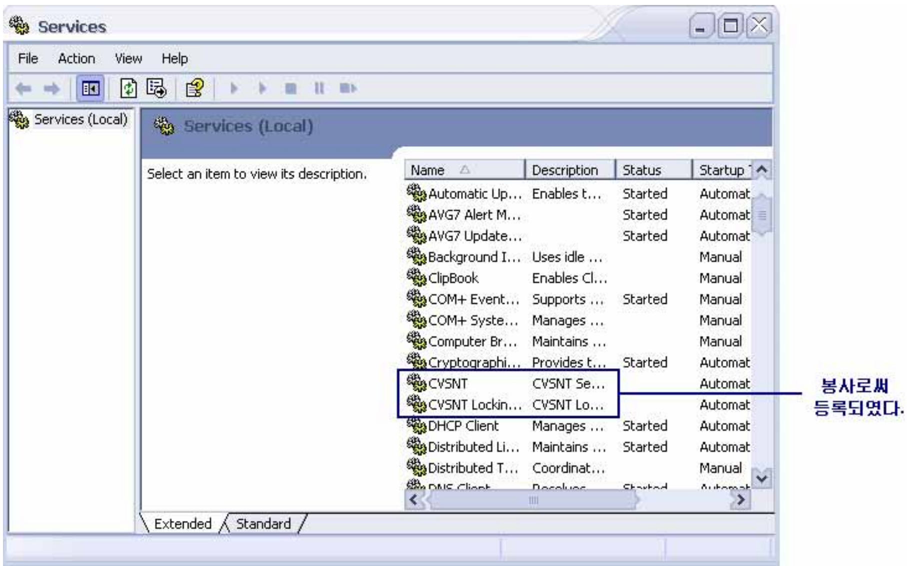


그림 9-7. Windows의 봉사기능으로서 등록

이것을 유효로 하기 위해 콤퓨터를 재기동한다.

9.2.3. 보관고작성

CVS봉사기를 사용하자면 우선 보관고를 작성하여야 한다. CVS의 보관고는 특별한 자료기지 등을 사용하는것은 아니다. 파일체계상의 보통 등록부를 보관고로서 설정할뿐이다. 여기서는 《c:\cvsrep》등록부를 사용한다.

Start차림표로부터 조종판(Control Panel)을 열어 classic보기로 절환하면 CVS for NT의 아이콘이 나타난다(그림 9-8). 이 아이콘을 두번 찰칵한다(그림 9-9).

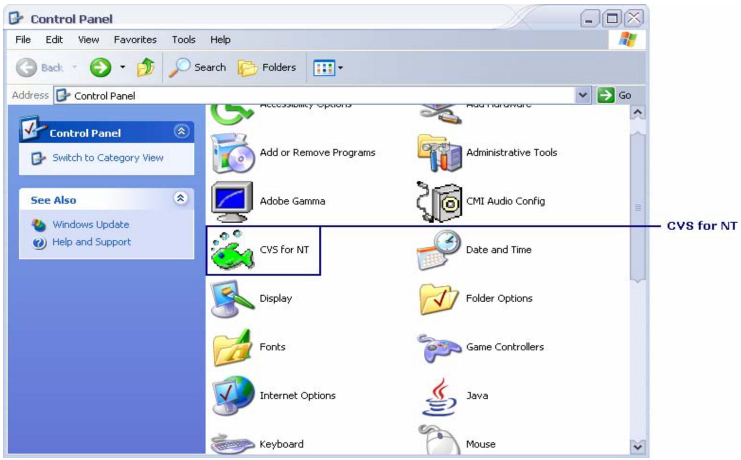


그림 9-8. CVS for NT아이콘

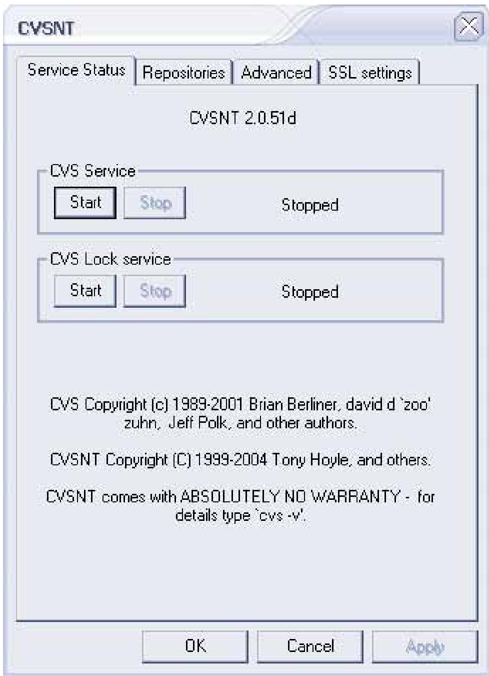


그림 9-9. CVS조종판

[repositories]태브를 찰칵한다(그림 9-10).

cvsnt는 여러개의 보관고를 관리할수 있지만 지금은 한개도 등록되여있지 않다. [Add]단추를 찰칵하여 [Location]란의 오른쪽 [⋯]을 찰칵하여 [folder ⋯]대화칸을 연다. 여기서 \cvs등록부를 선택하고 [Make New Folder]을 찰칵하여 [cvsrep]등록부를 작성해놓는다(그림 9-11). [OK]단추를 찰칵하면 [Edit repository]대화칸의 [Location]과 [Name]란에 이름들이 채워진다(그림 9-12).

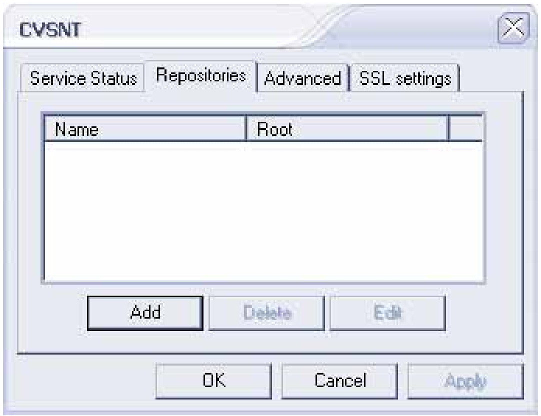


그림 9-10. 보관고

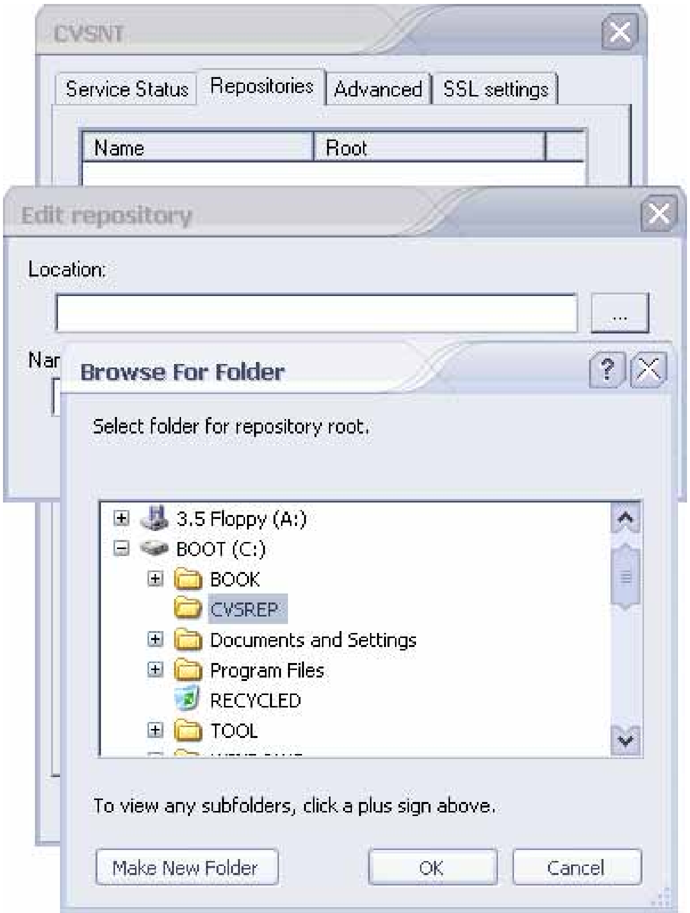


그림 9-11. 보관고등록부의 작성



그림 9-12. 보관고선택

실제로 cvs말단기가 이 봉사기에 접근할 때 [Name]란에 정의된 문자렬을 지정하면 된다. [OK]단추를 찰칵한다. \cvsrep등록부를 cvs보관고로서 초기화하겠는가를 묻는 대화칸이 현시된다. 초기화하지 않으면 사용할수 없기때문에 [Yes]단추를 찰칵한다. c:\cvsrep등록부가 보관고로서 설정되면 [Advanced]태브로 절환하여 [Pretend to be a Unix CVS version]검사칸을 선택한다. Eclipse는 낡은 판본의 cvsnt와는 정합이 잘 되지 않으므로 보관고와의 동기화가 잘 진행되지 않는다. 이 설정에 의해 cvsnt가 UNIX판 CVS경우에 비해 이러한 현상을 회피할수 있게 한다(그림 9-13).

우의 설정들을 진행하여 cvs봉사기로서 쓸수 있게 되였다. [OK]단추를 찰칵한다.

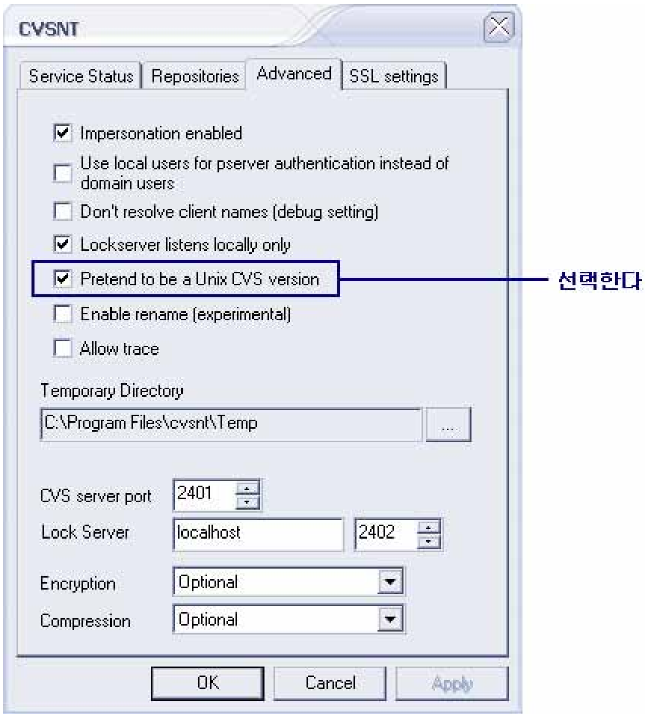


그림 9-13. Eclipse로부터 실행시키기 위한 조건선택

제3절. Eclipse를 사용한 CVS봉사기에로의 접근조종

9.3.1. CVS에서 사용되는 용어

표 9-1에 CVS를 리용할 때 사용되는 용어들에 대하여 간단히 종합하였다.

표 9-1. CVS에서 사용되는 용어

용 어	설 명
모 듈 (Module)	판본관리대상으로 되는 파일의 집합체이다. 한개 보관고안에 여러개의 모듈을 등록할수 있다. Eclipse에서는 1개 프로젝트에는 1개의 모듈이 대응된다.
들여넣기 (Import)	작업대상프로젝트(또는 파일)를 공유하기 위하여 보관고에 등록하는것을 말한다.
꺼내오기 (Checkout)	보관고로부터 모듈을 작업콤퓨터에 꺼내오는것을 말한다.
작업복사물	꺼내온 파일들을 말한다. 실제의 편집작업은 작업복사물에 대하여 진행한다. Eclipse에서는 보관고로부터 작업공간상으로 꺼내온 프로젝트가 작업복사물이다.
보관하기 (Commit)	작업복사물에 대하여 진행된 변경을 보관고에 등록하는것을 말한다. 이것에 의해 보관고상의 파일에 대하여 교정(Eclipse에서는 [개정]이라고 한다.)이 하나씩 진행된다.
갱신하기 (Update)	보관고의 최근 수정내용을 작업복사물에 반영시킨다.
태브 (Tab)	모듈의 어떤 특정한 시점에서 파일들에 붙인 이름을 말한다. 태브가 붙여진 어떤 시점에서의 모듈내용을 언제나 재현할수 있다.
줄기 (Trunk)	보관고안의 기본개발흐름선이다. 보통 처음에 판본관리대상을 줄기에 들여넣기하여 모듈을 작성하기때문에 꺼내오기, 편집, 보관하기 등 작업의 반복이 줄기상에서 진행된다. HEAD란 줄기의 앞머리를 나타내는 태브이다. 《HEAD=줄기》라고 생각해도 된다.
가지 (Branch)	줄기(Trunk)에 대한 가지선을 말한다. 새로운 기능을 추가하거나 오유의 수정 등은 가지상에서 진행하기때문에 줄기에 그 내용을 반영(줄기와 정합)하는 경우가 많다.

9.3.2. 다중사용자 환경

설치한 CVS봉사기(CVS보관고)에 Eclipse작업대로부터 접근조종하여 실제로 CVS를 사용해본다. 팀개발을 진행한다면 반드시 맞다드는 변경의 다투기나 정합에 대하여 아래와 같은 실례를 들어 보기로 하자.

이미 7장에서 사용한 SampleChap7프로젝트를 다중개발자가 공유하여 팀개발을 진행하도록 한다. 주요 개발자는 cvsuser01과 cvsuser02인 2명이다. 개발자는 각각 SampleChap7프로젝트를 자기의 콤퓨터상의 작업공간에 꺼내여 개발작업을 진행한다. 이러한 상황에서 2대의 콤퓨터(CVS봉사기 한대, Eclipse말단기 한대)와 1개의 Eclipse상에서 재현하는 방법에 대하여 설명한다.

1) 사용자등록

cvsuser01, cvsuser02라는 2명의 사용자를 등록한다.

표준설치된 cvsnt에서는 사용자를 Windows사용자그대로 리용하게 되여있기때문에 Windows에서 이 사용자들을 등록한다.

① [Start]→[Control Panel]을 찰칵하고 [User Accounts]를 찰칵한다.

② [Create a new account]단추를 찰칵한다.

③ [Type a name for the new account:] 마당에 《cvsuser01》이라고 입력하고 [next]단추를 찰칵한다(그림 9-14).

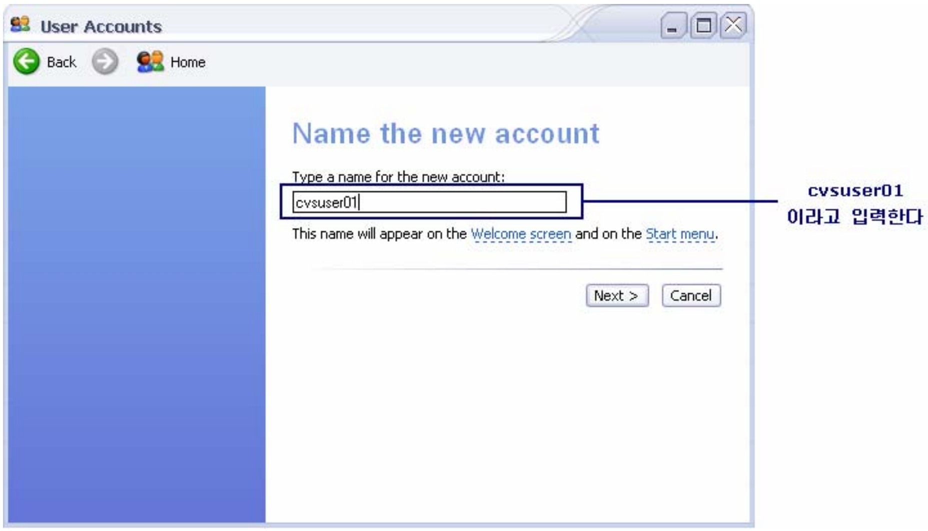


그림 9-14. 새로운 사용자이름 설정

④ [Computer Administrator]를 선택한 다음 [Create account]를 찰칵한다(그림 9-15).

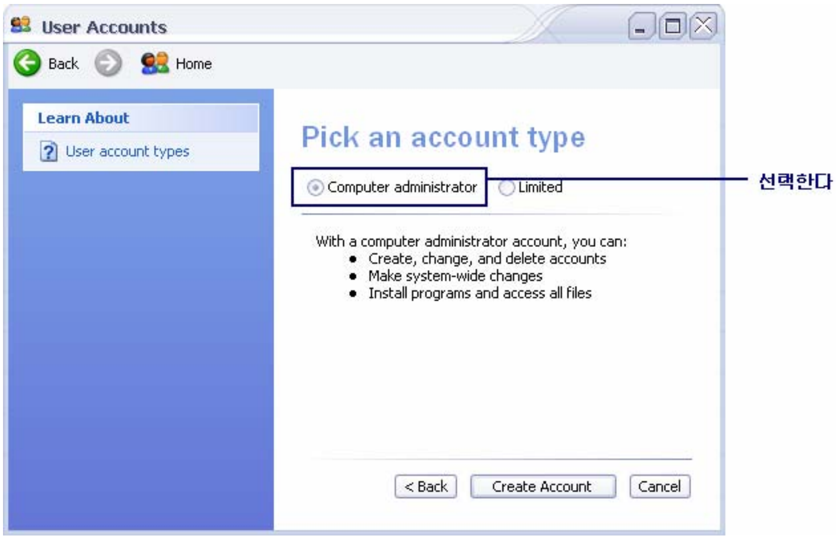


그림 9-15. 사용자부류의 선택

⑤ 작성한 사용자에 대해서 암호를 설정한다. 이자 만든 사용자를 찰칵한다.

⑥ [Create a password]단추를 찰칵한다.

⑦ 암호를 작성한다. [Type a new password]마당과 [Type the new password again to confirm]마당에 암호를 입력하고 [Create password]를 찰칵한다(그림 9-16).

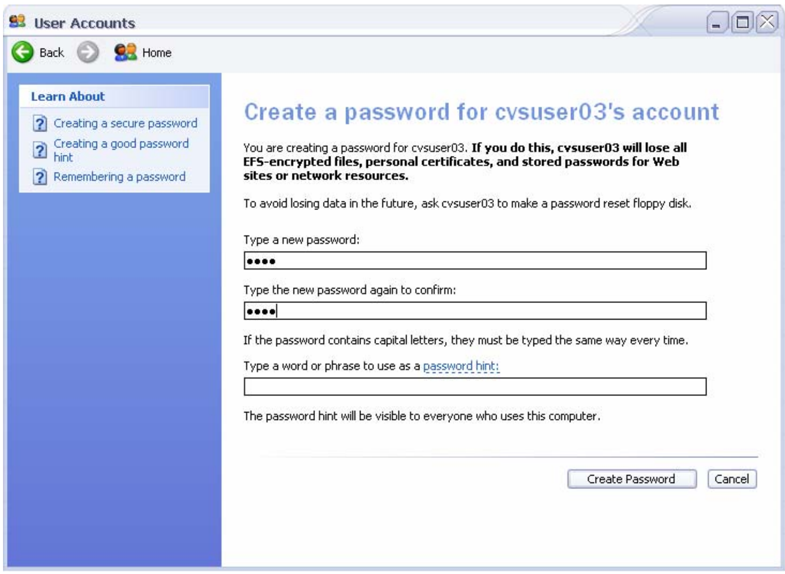


그림 9-16. 암호의 작성

사용자 cvsuser01의 작성이 완료되였다. 꼭같은 방법으로 cvsuser02도 만든다.

2) 보관고위치의 설정

다음 Eclipse에서의 설정을 진행한다. [CVS Repository Exploring]투시로 절환하여 cvsuser01, cvsuser02의 개개의 전용보관고위치를 작성한다.

보관고위치란 CVS보관고의 장소나 사용자등록에 관한 정보 등을 종합한것이므로 Eclipse에서 CVS보관고에로의 접근조종을 실현할 때 기준점으로 된다.

① 차림표띠에서 [Window]→[Open Perspective]→[Other…]를 선택하여 투시의 선택대화칸을 현시하고 [CVS Repository Exploring]를 선택한다.

② [CVS Repository Exploring]투시가 열리면 [CVS Repositories]보기창에서 오른쪽찰칵하여 펼쳐지는 차림표에서 [New]→[Repository Location…]를 선택한다.

③ [Add CVS Repository]대화칸을 연다. Eclipse작업대에서 CVS보관고를 호출조종할수 있도록 그 장소를 지정한다. 그림에서와 같이 정보를 입력한다(그림 9-17).

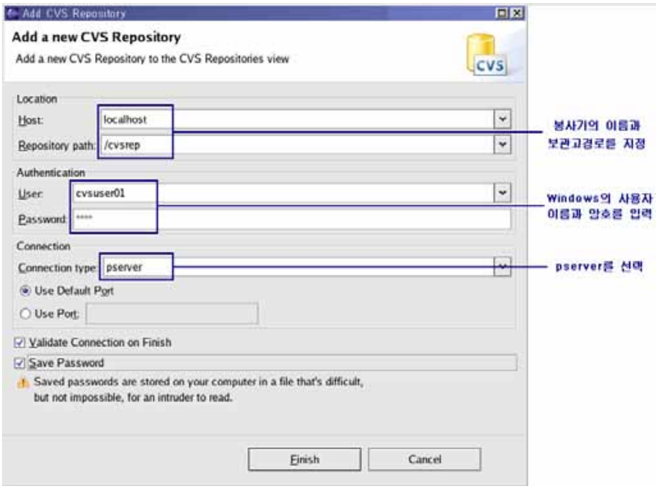


그림 9-17. CVS보관고추가대화칸

- Location(CVS보관고의 장소를 지정)

Host: CVS보관고가 있는 호스트콤퓨터이름을 지정한다. Windows용콤퓨터에 봉사기환경을 구축하였기때문에 그 콤퓨터의 망IP주소를 기입하는데 여기서는 《172.29.92.105》로 한다.

Repository path: CVS보관고가 있는 장소를 지정한다. CVSNT의 조종판에서 설정한 [Name]의 값을 사용한다. 여기서는 먼저 작성한 《/cvsrep》를 사용한다.

- Authentication(CVS보관고에서 호출조종하는 사용자정보를 지정)

cvsnt의 기정설정에서는 Windows에 등록가입할 때의 사용자인증구조를 리용하게 되여있다. 따라서 [User], [Password]에서도 Window에서의 설정을 그대로 사용한다.

여기에서는 사용자 《cvsuser01》을 지정한다.

- Connection(CVS보관고의 호출접근경우에 사용하는 통신규약을 지정)

Connection type: pserver, ext, extssh의 3가지가 사용가능한데 여기에서는 pserver를 지정한다. 포구설정은 [Use Default Port]으로 한다

Save password: 이것을 선택해놓으면 암호입력을 재촉함이 없이 넘어간다. 기업소내 국부망과 같은 안전한 환경에서 사용하고있다면 선택해놓는것이 편리하다. 다음 [Finish]단추를 찰각한다.

④ CVS보관고에로의 접속이 성공하면 그림 9-18과 같은 화면이 펼쳐진다.

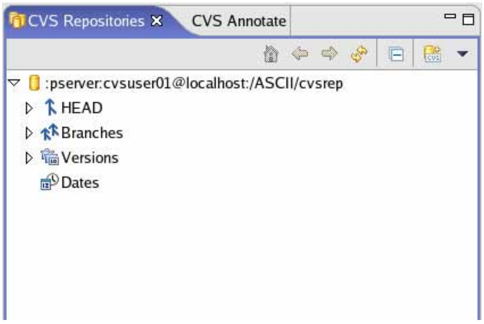


그림 9-18. 보관고추가완성

꼭같은 방법으로 cvsuser02보관고위치도 작성한다.

사용자이름만 내놓고 나머지는 지금까지 작성한 cvsuser01의 보관고위치설정과 꼭같은 내용이다(그림 9-19).

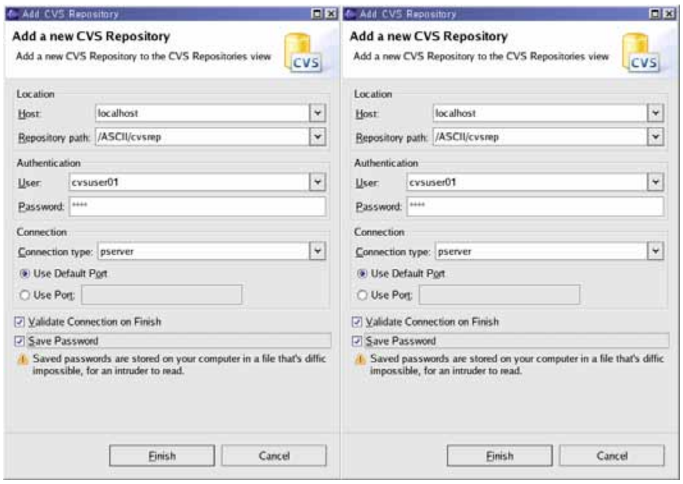


그림 9-19. 사용자이름만 다르게 설정

[CVS Repositories]보기에서 보면 그림 9-20과 같다.

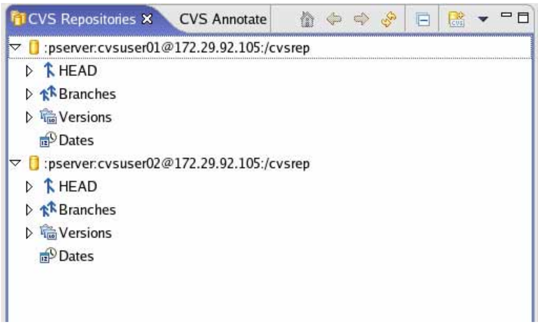


그림 9-20. 두개의 보관고위치

9.3.3. 프로젝트를 보관고에 등록

우선 처음으로 진행해야 할것은 이제부터 자기가 작업하는 프로젝트를 CVS보관고에 등록하는것이다. 여기에서는 SampleChap7프로젝트의 개발팀성원 cvsuser01이 개발하는 제일 초기코드를 보관고에 등록한다.

① SampleChap7프로젝트를 작업공간에 복사한다.

② Java투시의 패키지열람기에서 [SampleChap7]프로젝트를 오른쪽찰칵하고 내부차림표로부터 [Team]→[Share Project]를 선택한다.

③ 어떤 보관고에 대하여 프로젝트를 공유하는가를 지정한다.

여기에서는 처음에 작성한 cvsuser01용보관고를 사용한다(그림 9-21). [Next]단추를 찰칵한다.

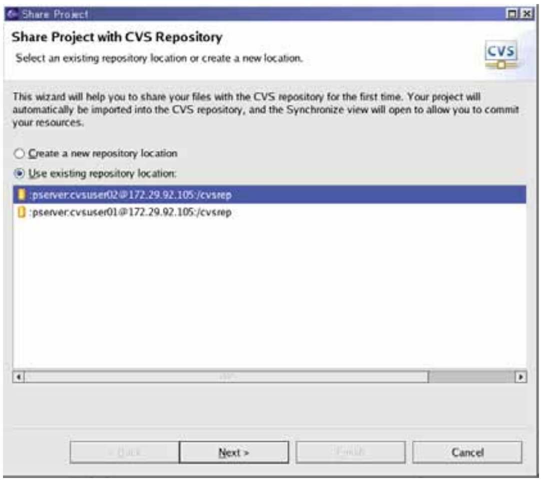


그림 9-21. 보관고선택

④ 공유하려는 프로젝트를 cvs보관고에 대하여 어떤 이름으로 등록하는가를 지정한다.

여기에서는 [CVS Repository Exploring]투시에 현시된 이름으로 한다. 보통 사용자가 작성한 프로젝트이름을 그대로 등록이름으로 하는 경우가 많지만 특별히 이름을 지정할수도 있다. 여기에서는 프로젝트이름으로 모듈이름을 사용한다(9-22).

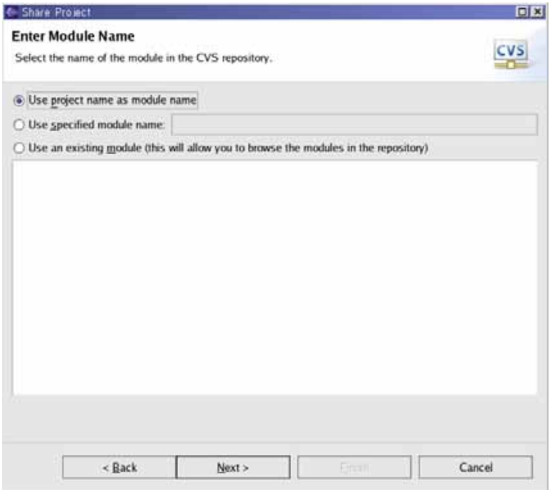


그림 9-22. 모듈이름의 입력

⑤ [Next]단추를 찰칵하면 CVS보관고에 련결하는 작업이 시작되고 그후 확인화면이 나타난다. ▷단추를 찰칵하여 계층구조를 전개시켜 보면 프로젝트파일이 보관고안의 등록대상으로 되여있는것을 알수 있다(그림 9-23).

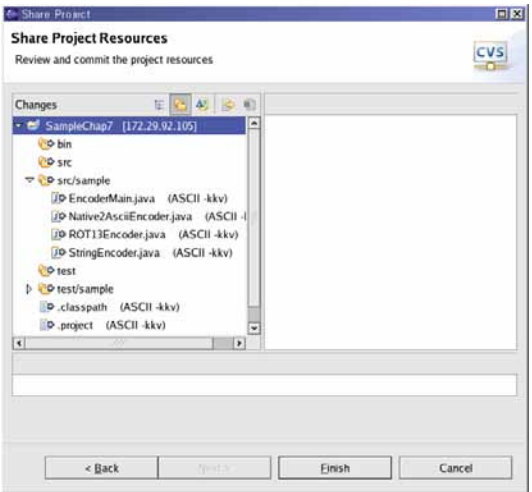


그림 9-23. 보관고안에 등록된 프로젝트의 미리보기

⑥ [Share Project]대화칸에서 프로젝트이름을 오른쪽찰칵하여 펼쳐지는 문맥차림표로부터 [Commit]를 선택한다. 그러면 그림 9-24와 같은 대화칸이 나타난다.

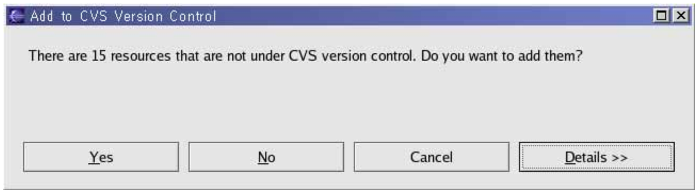


그림 9-24. 보관의 미리보기

[Yes]단추를 찰칵하면 보관에 대한 설명입력대화칸이 현시되는데 여기서 입력하는 설명문은 후에 변경리력을 열람할 때에 표시되게 된다. 이 보관을 진행하는 목적에 대한 간단한 설명을 입력하고 [OK]단추를 찰칵한다(그림 9-25).

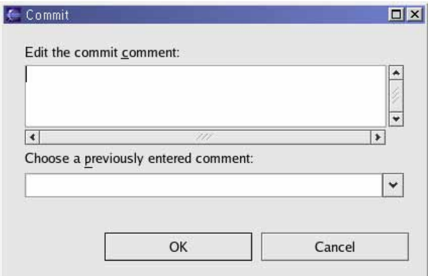


그림 9-25. 보관에 대한 설명입력칸

《언제》, 《어느 파일을 보관하였는가》는 CVS가 기록해두기때문에 사용자가 입력할 필요가 없다. [OK]단추를 찰칵하면 그림 9-26과 같은 화면이 나타난다. [Finish]단추를 찰칵하여 완료한다.

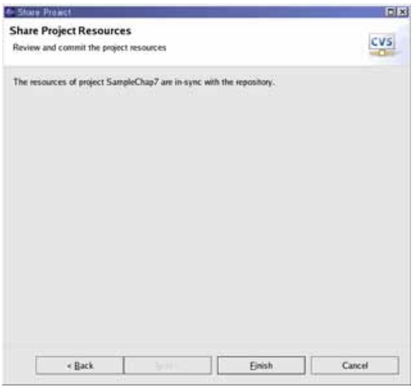


그림 9-26. 보관의 완료

⑦ 작업공간상의 프로젝트에 어떤 변화가 일어나는가 확인해본다.

Java투시안의 [Package Explorer]보기에서 공유된 프로젝트(SampleChap7)를 오른쪽찰칵하여 나타나는 문맥차림표에서 [property]를 선택하면 화면안쪽에 [CVS]라는 항목이 추가되여있고 (HEAD)가 붙어있는것을 알수 있다(그림 9-27).

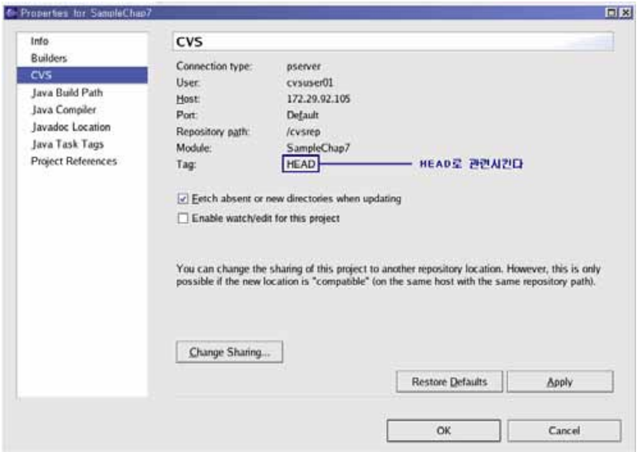


그림 9-27. 프로젝트의 속성

⑧ 지금 상태를 CVS보관고측에서 다시 한번 본다.

[CVS Repository Exploring]투시로 절환하고 [CVS Repository]보기에서 CVS 보관고위치의 ▷단추를 찰칵하여 전개한다. [HEAD]→[SampleChap7]을 전개해본다. 그러면 그림 9-28과 같은 결과가 나타난다.

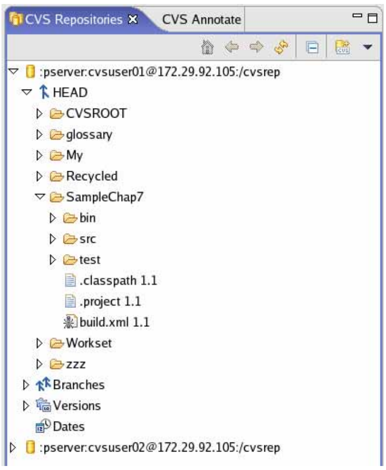


그림 9-28. 보관고에 등록된 프로젝트

프로젝트가 무사히 보관고에 등록된것을 볼수 있다. 파일의 옆에 있는 수자는 [교정번호]이다. 이것은 파일의 판본을 표시하며 초기값은 1.1이다.

9.3.4. 보관고로부터 꺼내오기

앞절에서는 SampleChap7프로젝트를 보관고에 등록하고 팀개발을 할수 있는 준비를 갖추었다.

이 절에서는 개발팀의 매 성원이 프로젝트를 보관고로부터 꺼내서 개발을 진행하는 방법에 대하여 설명한다. 여기서는 SmapleChap7프로젝트의 개발팀성원 cvsuser02가 SampleChap7프로젝트를 보관고로부터 꺼내오는 실례를 보기로 한다. 실제로 팀개발에서는 cvsuser02가 자기의 콤퓨터의 작업공간으로 꺼내오기하지만 여기서는 한대의 콤퓨터상에서 한개의 Eclipse작업공간에서 팀개발을 진행한다는데 주의를 돌려야 한다.

우선 cvsuser01이 SampleChap7을 보관고에 등록한것과 같은 작업공간상에 cvsuser02가 보관고장소에서부터 SampleChap7프로젝트를 꺼내온다. 일반적인 꺼내오기를 진행하면 프로젝트이름이 충돌하기때문에 《별명 꺼내오기》를 진행한다.

① [CVS Repository Exploring]투시로 절환하여 cvsuser02의 보관고위치의 [HEAD]를 전개하여 [SampleChap7]을 오른쪽찰칵하고 [Check Out As(별명 꺼내오기)]를 선택한다.

② [Project Name]마당에 [SmapleChap7_02]라고 입력하고 [Finish]단추를 찰칵한다(그림 9-29).

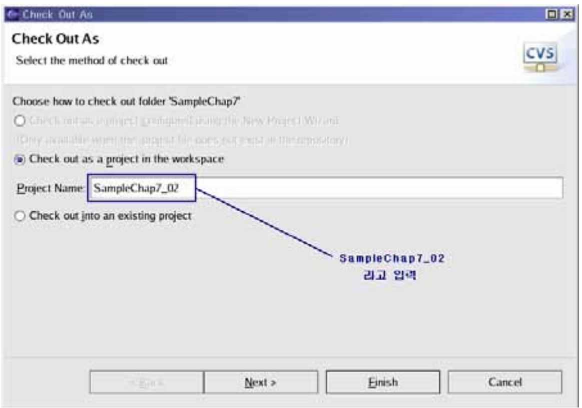


그림 9-29. Check Out As

이렇게 함으로써 cvsuser02에 의해 프로젝트가 작업공간상에 현시되였다.

③ [Java]투시로 절환하여 작업공간을 확인한다(그림 9-30).

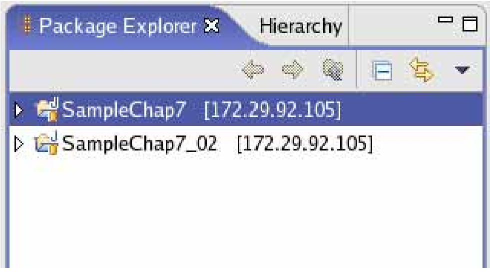


그림 9-30. 두개의 프로젝트

2개의 프로젝트를 정확히 꺼내왔다.

이것으로 SampleChap7에 대한 작업은 cvsuser01사용자로서, SampleChap7_02에 대한 작업은 cvsuser02사용자로서 각각 보관고에 보관되고 한대의 콤퓨터상에서 하나의 Eclipse로 다중사용자에 의한 팀개발환경이 마련되였다.

9.3.5. 표식자장식의 설정

보관고로부터 프로젝트를 꺼내올 때《표식자장식》이라는 기능을 리용해보기로 한다. 이것을 리용하면 CVS보관고와 관련된 작업공간상의 자원에 여러가지의 CVS관련정보(자료보관이름, 가지이름, 교정번호 등)를 표시할수 있다.

표식자장식을 설정하자면 [Window]→[Preferences]를 선택하여 [Preferences]대화칸을 현시한다. [Workbench]→[Label Decorations]을 선택하여 [Available label decorations]의 [CVS]를 선택한 다음 [OK]단추를 찰칵한다(9-31).

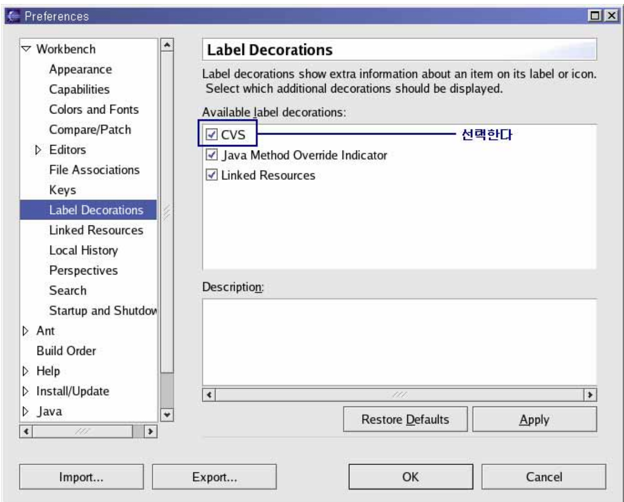


그림 9-31. 표식자장식의 설정

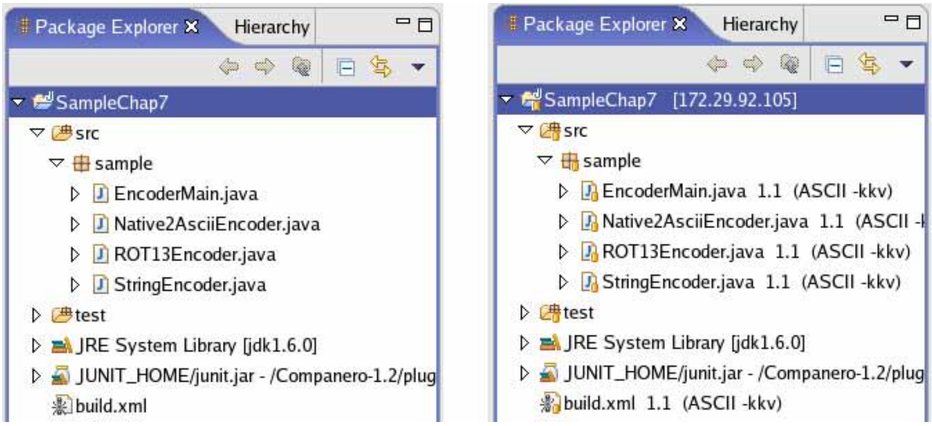


그림 9-32. 표식자장식 없음(왼쪽), 표식자장식 있음(오른쪽)

제4절. 보관고와 동기맞추기

개발자는 자기의 작업공간에 꺼내온 프로젝트에 대하여 작업을 진행한다. 작업의 결과를 팀에서 공유하자면 개발자들이 개별적으로 진행한 변경을 보관고에 등록하여야 한다. 이것을 《보관하기》라고 한다. 또한 반대로 다른 개발자가 보관한 변경내용을 자기의 작업공간에로 불러들일 필요도 있다. 이렇게 하는것을《갱신하기》라고 한다. Eclipse에서는 이것을 [Team synchronization]투시에서 일괄적으로 취급한다.

9.4.1. 발신변경

여기서는 SampleChap7프로젝트의 개발팀성원인 《cvsuser01》가 Native2AsciiEncoder.java에 변경을 가한다. 실례를 들어보기로 하자.

cvsuser01은 이 변경을 보관고에 보관한다.

— Native2AsciiEncoder.java의 변경

여기서는 cvsuser01이 Native2AsciiEncoder.java의 클라스설명부분에서 @Version태그에 CVS의 열쇠어인 《$Revision$》을 써넣는다.

우의 례에서 본 《$Revision$》은 무슨 의미를 담고있는가.

CVS에서는 열쇠어를 원천파일에 써놓으면 보관할 때 교정번호를 자동적으로 써넣고 《$Revision:1.2$》라고 표현한다. 우의 례와 같이 @Version태그에 《$Revision$》을 써넣으면 API문서를 생성할 때 클라스의 판본으로서 CVS교정번호가 표시되기때문에 편리하다.

그밖에 자주 리용되는 열쇠어를 다음의 표에 주었다.

표 9-2. 주요 열쇠어

열 쇠 어	설 명
$Author$	보관을 진행한 사용자이름이다. $Author: cvsuser01$
$Date$	보관이 진행된 시간이다. $Date: 2005/09/19 18:30:45$
Id.	파일이름, 교정, 날자시간 등을 써넣는것만으로 모든 정보가 기록된다. $Id:Sample.java,v 1.3 2005/09/19 18:30:45 cvsuser01 Exp$
Log	파일이름, 교정, 보관날자, 보관사용자이름이 추가되여 《보관하기》통보문의 리력으로 치환된다. Log와 행의 앞머리사이에 있는 문자(아래의 례에서는《*》)가 다음행의 앞머리에도 입력되기때문에 아래의 례와 같은 설명문만으로 Log를 서술해도 착실히 리력이 남는다. /** * $Log: Sample.java,v $ * Revision 1.3 2005/09/19 18:30:45 cvsuser01 * 열쇠어 치환시험 * * Revision 1.2 2005/09/19 18:15:45 cvsuser01 * 오유 ID 12345 를 수정 * * Revision 1.1 2005/09/19 18:10:45 cvsuser01 * *** 기록통보가 없음 *** * */

— 차이나는 부분의 확인

패키지열람기에서 프로젝트이름을 오른쪽찰칵하여 [Team]→[Synchronize with Repository]를 찰칵한다. 투시절환확인대화칸이 나타나면 [OK]단추를 찰칵하여 [Team Synchronizing]투시에로 절환한다(그림 9-33).

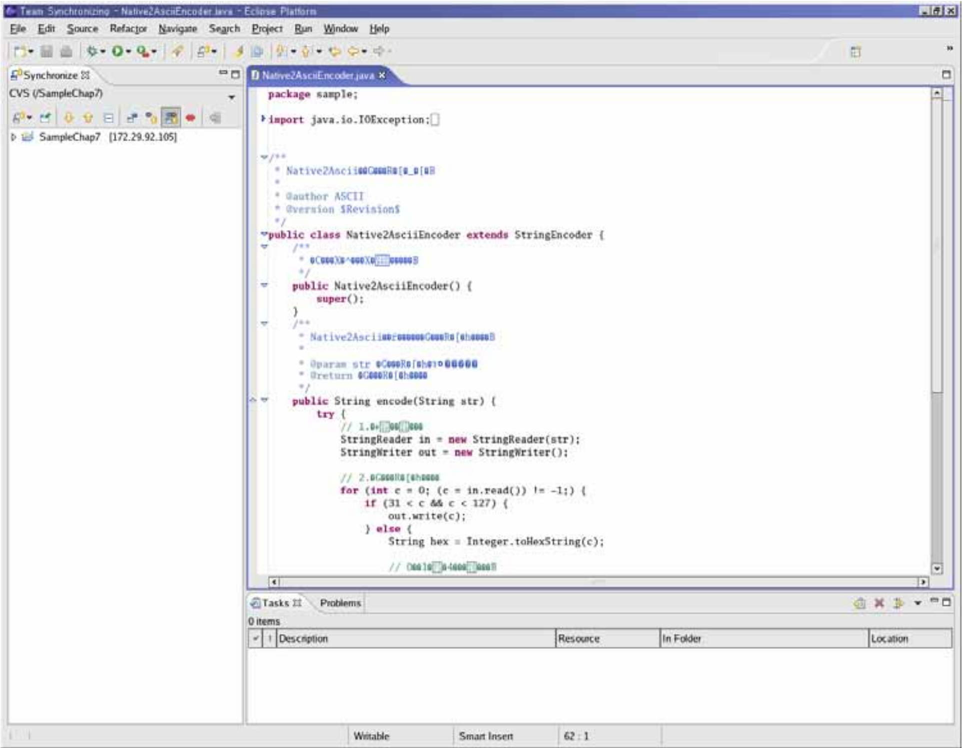


그림 9-33. [Synchronize]투시

기본은 [Sychronize]보기이다. 이 보기에서는 작업공간과 보관고의 상태에서 어떤 차이가 있는가를 비교해볼수 있다.

프로젝트를 오른쪽찰칵하여 [Expand All]를 선택하면 [Native2AsciiEncoder.java]에 변경이 가해졌는가를 알아볼수 있다. 회색의 오른쪽화살표식은 작업공간의 파일에 변경이 가해졌다는것을 표시하고있다(그림 9-34).

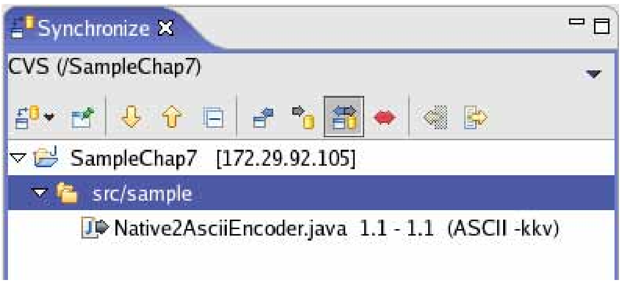


그림 9-34. [Synchronize]의 보기

파일을 두번찰칵하면 변경내용을 확인할수 있다(그림 9-35).

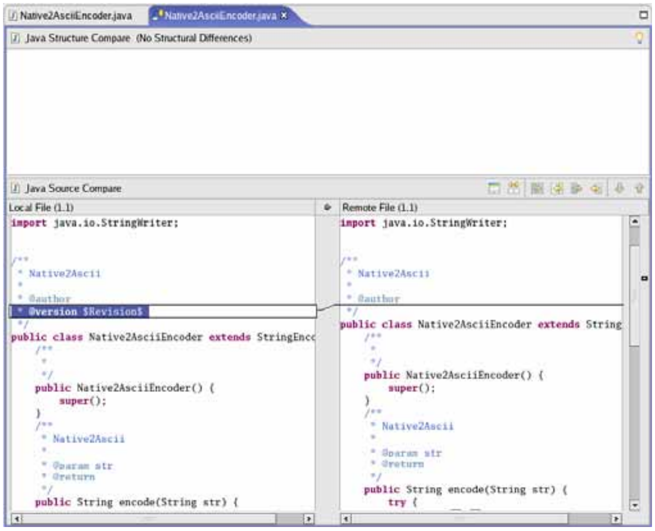


그림 9-35. 작업공간과 보관고와의 차이 비교

① Java구조체의 비교

국부파일(작업공간), 원격파일(보관고)의 2개 파일의 차이가 표시된다. 이 례에서는 설명문만 변경하였으므로 구조상 차이를 찾아볼수 없다.

② 국부파일

작업공간의 파일내용이다. 파일이름옆의 《1.1》이란 수자는 이 파일이 보관고상의 교정1.1을 꺼내온것임을 표시한다.

③ 원격파일

보관고의 파일내용이다. 《1.1》이라는 수자 역시 교정번호이며 이 파일의 최신판이 교정1.1이라는것을 표시하고있다.(결국 초기상태 그대로라고 말할수 있다.) 이것으로 cvsuser01은 자기의 작업공간의 파일과 보관고의 파일의 차이를 확인할수 있다.

— 보관고에 보관

다음에 이 국부적인 변경을 보관고에 보관한다. [Synchronize]보기의 [Outgoing Mode(전체 발신변경을 저장한다)]단추를 찰칵한다(그림 9-36).

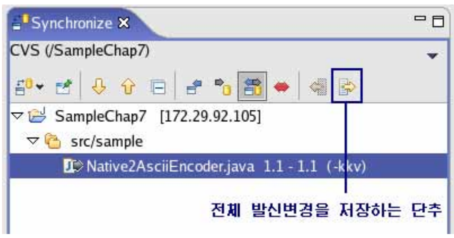


그림 9-36. 발신변경의 보관

보관설명입력대화칸이 표시되기때문에 설명을 입력하고 [OK]단추를 찰칵한다. 이것으로 동기화가 완료되고 작업공간과 보관고의 내용은 완전히 일치되였다. 그리고 Java투시로 절환해본다. [Package Explorer]보기를 보면 명확히 Native2AsciiEncoder.java가 1.2로 되여있는것을 확인할수 있다(그림 9-37).

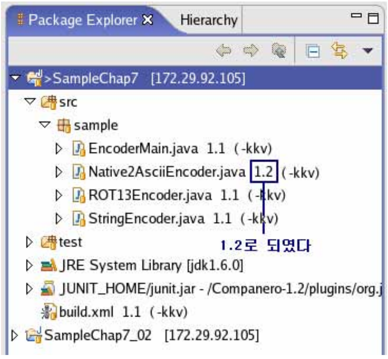


그림 9-37. 교정번호의 증가

9.4.2. 착신변경

여기서는 cvsuser01이 진행한 변경을 cvsuser02이 자기의 작업공간으로 끌어들이는 경우에 대하여 보기로 한다. Cvsuser02의 작업이므로 [SampleChap7_02]프로젝트에 대하여 진행한다.

— 변경내용의 확인

[SampleChap7_02]프로젝트를 오른쪽찰칵하여 [Team]→[Synchronize with Repository]를 찰칵한다. 투시로 절환하여 확인대화칸이 펼쳐진 경우 [OK]단추를 찰칵한다. 푸른색의 왼쪽화살표는 보관고에 새로운 변경이 있다는것을 알려준다. 교정번호가 《1.1-1.2》와 같이 2개 나란히 표시되여있는데 왼쪽은 작업공간에서 파일의 교정번호이고 오른쪽은 보관고에서 파일의 최신교정번호이다(그림 9-38).

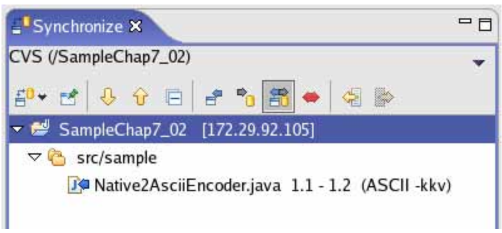


그림 9-38. 착신변경의 확인

이 파일을 두번 찰칵하여 변경내용을 확인해본다(그림 9-39).

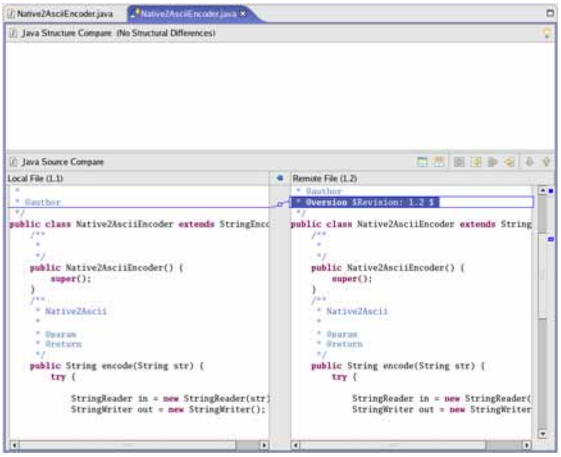


그림 9-39. 변경내용의 확인

클라스설명에 @version태그가 추가되여있는것을 볼수 있다. cvsuser01에 의하여 보관고에 보관될 때 $Revision$열쇠어가 [$Revision: 1.2$]이라고 교정번호를 포함하는 형식으로 보관되였기때문이다.

— 갱신하기(보관고의 내용 끌어들이기)

보관고의 최신내용을 작업공간으로 끌어들인다. [Synchronize]보기에서 프로젝트를 오른쪽찰칵하고 [Refresh]를 찰칵한다. [Java]투시로 절환하여 변경내용이 들어온것을 확인해본다(그림 9-40).

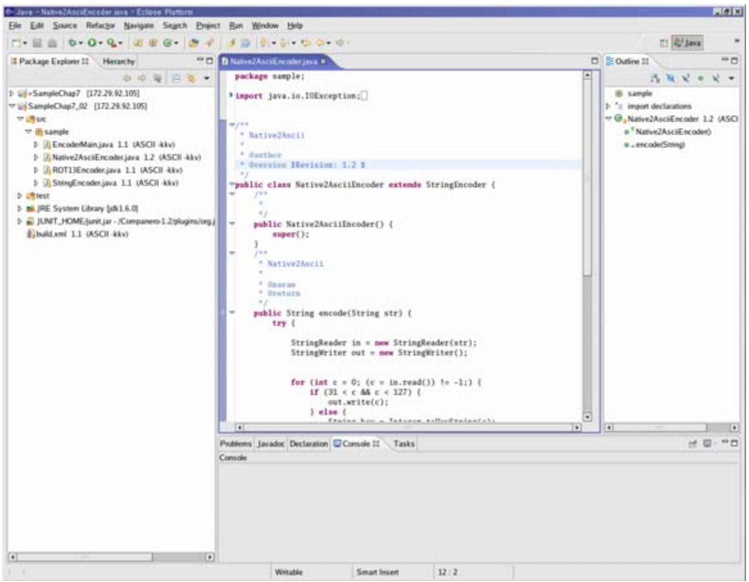


그림 9-40. cvsuser01에 의한 변경이 cvsuser02의 내용에 반영

9.4.3. 변경의 다투기

여기서는 다음의 경우에 대하여 상정한다.

cvsuser01은 Native2AsciiEncoder.java에 변경을 가하고 보관고에 보관하였다. cvsuser02는 그것을 모르고 비슷한 변경을 꼭같은 파일에 가하고 보관하려고 한다. 이와 같이 여러 성원이 파일의 꼭같은 부분을 동시에 변경시키려는것을 《변경의 다투기》라고 한다. 여기서는 다투기가 어떻게 일어나며 그것을 어떻게 해소시키겠는가에 대하여 설명한다.

— cvsuser01의 변경

cvsuser01은 Native2AsciiEncoder.java의 클라스설명을 편집하고 아래와 같이 @author태그에 자기의 사용자이름을 기입하고 보관하였다.

```
/**
 * Native2Ascii방식의 부호화기
 *
 * @author cvsuser01
 * @version $Revision: 1.2$
 */
```

— cvsuser02의 변경

cvsuser01이 진행한 변경을 알수 없는 cvsuser02는 우연히도 꼭같은 Native2AsciiEncoder.java에 대하여 비슷한 변경을 해버린다. 즉 클라스설명을 편집하고 @author태그에 자기의 사용자이름을 기입한다. 이 두 사람은 서로 《이것은 내가 쓴것이다.》라고 주장할것이다.

```
/**
 * Nantive2Ascii방식의 부호화기
 *
 * @author cvsuser02
 * @version $Revision: 1.2$
 */
```

cvsuser02는 이 변경을 보관하려고 [Synchronize with Repository]를 선택하였다. 같은 행을 매개 사용자가 변경한 자리는 비교편집기에서 붉은색표식이 붙고 다투기가 진행되고있다는것을 나타낸다(그림 9-41).

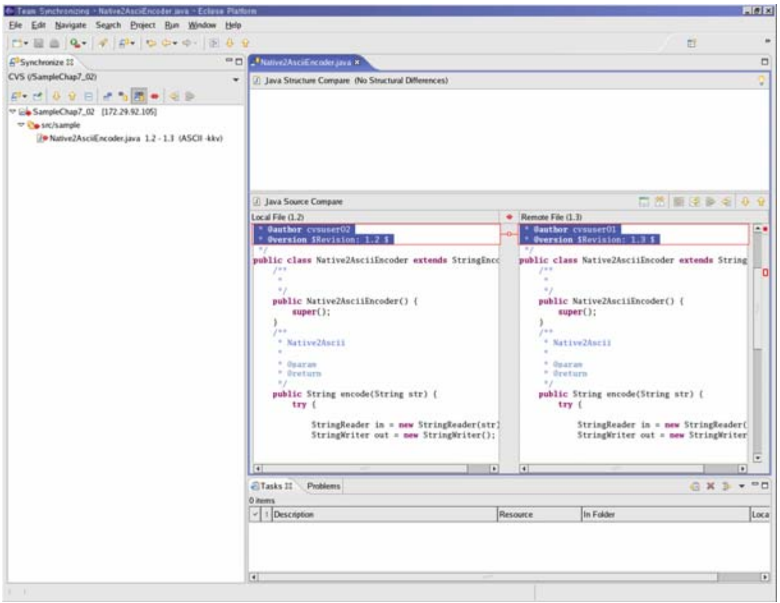


그림 9-41. 변경의 다투기

— 다투기의 해소

[Synchronize with Repository]를 선택하여 다투기상황에 직면한 개발자(이 례에서는 cvsuser02)에게는 먼저 해야 할것이 있다. 다투기상대(이 례에서는 cvsuser01)와 상론하여 정확한 코드를 결정하는것이다.

그 결과에 의해

경우1: 상대방의 코드를 채용하려는 경우

[Sychronize]보기에서 프로젝트를 오른쪽찰칵하여 현시되는 문맥차림표로부터 [Override and Update]을 선택하면 보관고의 내용으로 작업공간의 내용이 갱신된다.

경우2: 자기의 코드를 채용하려는 경우

[Sychronize]보기에서 프로젝트를 오른쪽찰칵하여 현시되는 문맥차림표로부터 [Override and Commit]를 선택하면 작업공간의 내용이 보관고에 보관된다.

경우3: 쌍방의 코드를 정합하려는 경우

비교편집기에서 쌍방의 코드의 차이점을 비교하면서 정합할수 있으며 보통 Java편집기에서 원천을 편집하고 보관한다.

여기에서는 cvsuser01과 cvsuser02의 상론결과 다음과 같이 하기로 합의하였다고 하자.

이 클라스는 cvsuser01과 cvsuser02의 공동작이므로 @author태그를 하나 추가하여 쌍방의 사용자이름을 렬거하기로 한다. 이 합의는 쌍방의 코드를 정합하는 《경우3》에 해당된다고 말할수 있다. [Java]투시로 절환하여 Native2AsciiEncoder.java를 아래와 같이 편집한다.

```
/**
 * Nantive2Ascii방식의 부호화기
 *
 * @author cvsuser01
 * @author cvsuser02
 * @version $Revision: 1.2$
 */
```

여기서 작업공간의 Native2AsciiEncoder.java는 합의에 따라 보관고에 보관하여야 할 내용으로 되였다. 다음에 이것을 보관하는 작업을 보기로 한다. [Ctrl]+[S]로 Native2AsciiEncoder.java를 보존한다.

— 보관하기

이상으로 파일의 내용으로서는 다투기가 해소되였다. 그러나 Eclipse는 아직 그것을 인식하지 못한다. [Sychronize]보기에서 [Native2AsciiEncoder.java]를 오른쪽찰칵하여 현시되는 문맥차림표로부터 [Mark as Merged]을 선택한다. 그러면 [Native2AsciiEncoder.java]는 정합에 따라 다투기가 해소되였다고 확인되고 다투기정합이 발신변경표식으로 변한다(그림 9-42).

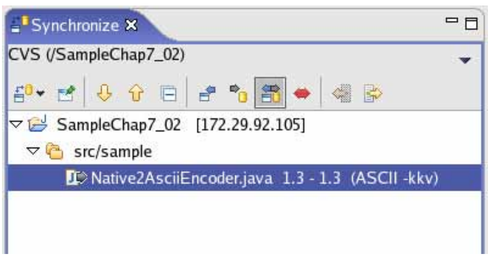


그림 9-42. 변경의 다투기가 해소되고 발신변경표식으로 변한다

마지막으로 진행된 변경을 보관고에 보관한다.

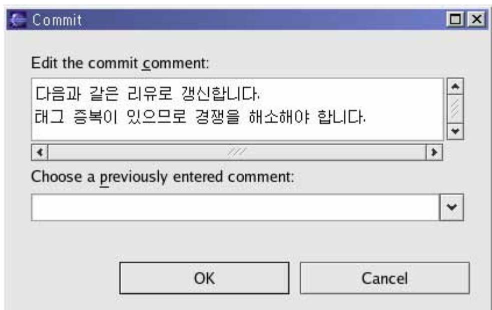


그림 9-43. 해당한 설명문작성

— cvsuser01측을 갱신

cvsuser02가 다투기를 해소한 결과를 보관하였다. 이것으로 Native2AsciiEncoder.java는 한개 교정이 더 진행되였다. cvsuser01의 프로젝트는 이 최신교정을 받아들일 필요가 있다.

이번에는 [Team Synchronizing]투시를 리용하지 않고 간단히 갱신을 진행해본다. [Java]투시로 절환하여 [Package Explorer]보기에서 [SampleChap7]프로젝트를 오른쪽찰칵하여 [Update]를 선택한다. [Update]대화칸이 표시되면 [Finish]단추를 찰칵한다(그림 9-44).

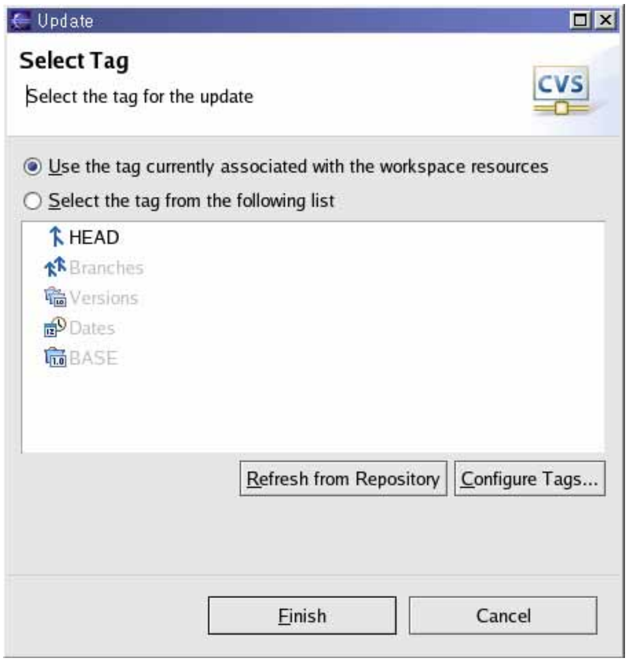


그림 9-44. [Update]대화칸

이것으로 cvsuser01의 SampleChap7프로젝트는 최신상태로 되였다.

제5절. 태그(판본)

CVS에서는 보관고에 첫 파일이 보관될 때 《1.1》이라는 교정번호가 붙는다. 그리고 파일을 갱신하면 《1.2, 1.3 … 》라고 교정번호는 올라간다. 이 교정번호에 의해 이전의 교정번호내용을 얻거나 2개의 교정번호사이의 차이를 명백히 할수 있다. 그러나 이 《교정번호》는 어디까지나 개개의 파일에 붙은 번호이다. 프로젝트에는 보통 여러개의 파일이 있고 개발이 진행되는데 따라서 개개 파일의 교정번호는 제가끔 되여있다. 그러므로 프로젝트의 임의의 한 시점에서 전체 파일에 대하여 교정번호가 얼마였는가를 기억해두어야 한다. 이것은 매우 불편하다. 그래서 CVS에서는 《태그》라는 개념을 도입하는데 이것은 여러개의 파일묶음에 대한 판본번호이다.

그림 9-45에 태그의 모양을 주었다. 5개의 파일로 되여있는 프로젝트에 《v1_0》과 《v2_0》인 2개의 태그가 붙어있다. 이렇게 태그이름들을 지정하는것으로써 프로젝트전체에 대한 《과거의 어떤 시점》에서의 상태를 정확히 파악할수 있다.

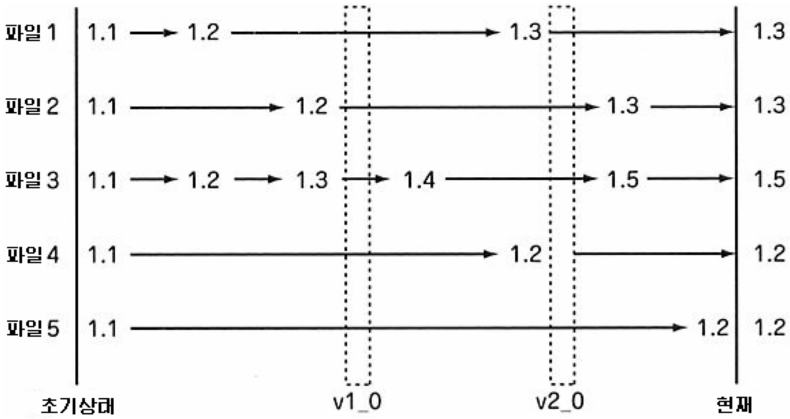


그림 9-45. 태그의 모양

9.5.1. 프로젝트에 태그를 붙인다.

여기서는 SampleChap7프로젝트의 개발팀성원인 《 cvsuser01 》이 현재의 SampleChap7프로젝트에 태그[v1_0]을 붙이는 경우에 대하여 보기로 한다.

① [Package Explorer]보기에서 프로젝트를 오른쪽찰칵하고 내부차림표로부터 [Team]→[Tag as Version]을 선택한다.

② [Tag Resources]대화칸에서 태그(판본)이름을 입력하고 [OK]단추를 찰칵한다(그림 9-46).

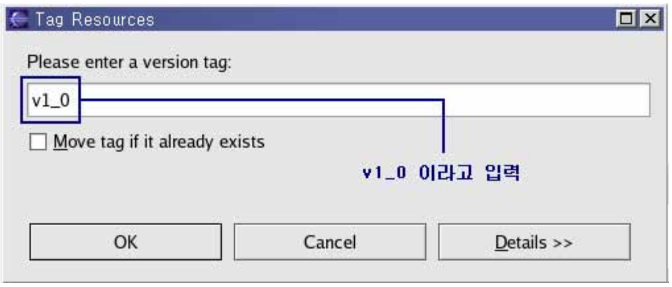


그림 9-46. 판본 이름을 입력

이것으로 태그(판본)를 붙이는 작업을 완료한다.

③[CVS Repository Exploring]투시에서 확인해본다.

[CVS Repository Exploring]투시로 절환하여 [CVS Repository]보기의 [Version]을 찰칵하여 전개한다(그림 9-47).

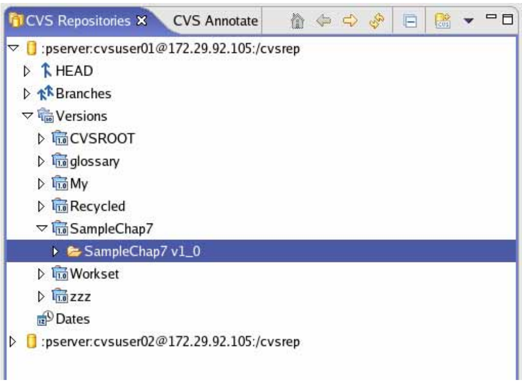


그림 9-47. 판본확인

보관고안에 판본이 작성되여있는것을 볼수 있다.

9.5.2. 판본을 지정하여 꺼내오기

프로젝트의 과거상태를 참조하려는 경우 판본을 지정하여 꺼내오면 판본을 붙인 시점에서의 프로젝트내용을 얻을수 있다.

① 꺼내려는 판본을 선택

[CVS Repository Exploring]투시로 절환하여 [CVS Repository]보기의 [Version]을 찰칵하여 전개한다.

② 꺼내려는 판본을 선택하여 오른쪽찰칵하고 내부차림표로부터 [Check out As]를 선택한다(그림 9-48).

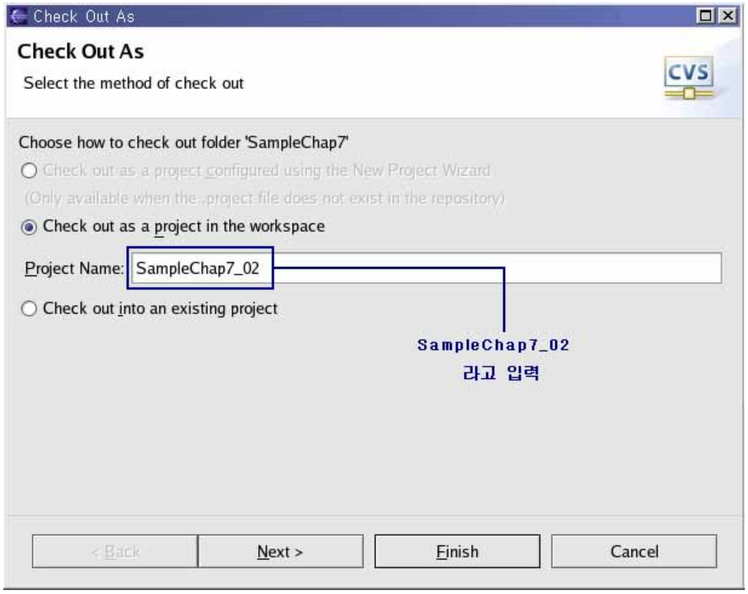


그림 9-48. 프로젝트이름 설정

[Import]를 선택했을 때 작업공간상에 프로젝트가 이미 존재하는 경우에 대해서는 우에서 서술되였기때문에 이 례에서는 [Check out As]를 사용한다. [Finish]단추를 찰칵하면 꺼내오기가 진행된다.

③확인해본다.

[Java]투시로 절환하여 [Package Explorer]보기에서 확인해본다(그림 9-49). 판본(태그)이름이 표식자장식으로서 되여있는것을 볼수 있다.

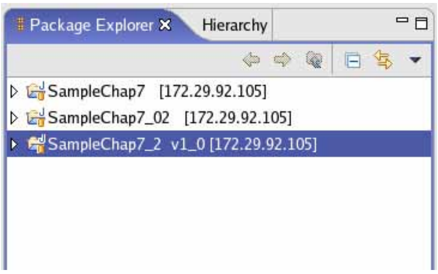


그림 9-49. 판본지정으로 꺼내온 프로젝트

주의: 즉 판본을 지정하여 꺼내오기한 프로젝트는 Eclipse상에서 편집할수 있지만 보관고에 보관하는것은 불가능하다. 《보관하기》조작자체는 선택할수 있지만 그림 9-50과 같은 오유가 나와 실패한다.



그림 9-50. 판본붙인 프로젝트에 대하여 보관실패

과거판본의 프로젝트내용으로 되돌아가서 개발을 계속 진행하려는 경우에는 [9.5.4. 과거의 판본에로의 되돌아가기]를 참조한다.

9.5.3. 판본을 지정하여 비교

판본을 지정하여 프로젝트의 내용을 비교할수 있다.

— 작업공간과의 비교

작업공간의 프로젝트를 보관고의 특정의 판본과 비교할수 있다.

여기서는 SampleChap7프로젝트의 개발팀성원인 cvsuser01이 판본[v1_0]의 공개후 Native2AsciiEncoder.java의 클라스설명문을 편집한 경우 그 프로젝트를 [v1_0]과 비교하는 실례를 보기로 한다.

편집내용은 Native2AsciiEncoder.java클라스의 설명문에 아래와 같은 하나의 문장을 추가한것이다.

```
/**
 * Nantive2Ascii방식의 부호화기
 * Nantive2Ascii설명문과 꼭같이 문자렬을 유니코드화한다.
 *
 * @author cvsuser01
 * @author cvsuser02
 * @version $Revision: 1.4$
 */
```

① Java투시의 [Package Explorer]보기에서 프로젝트를 오른쪽찰칵하여 내부차림표로부터 [Compare with]→[Another Branch or Version]을 선택한다.

② 비교대상판본을 선택

[Compare With Branch or Version]대화칸에서 비교대상판본을 선택한다(그림 9-51).

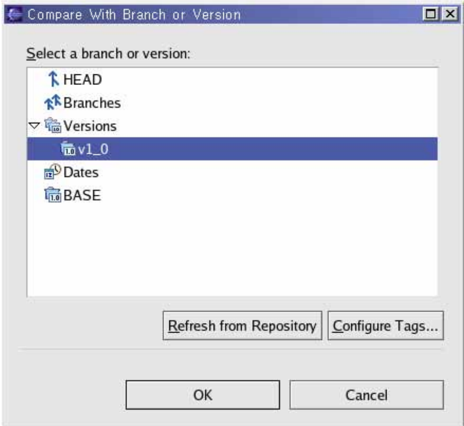


그림 9-51. [Compare With Branch or Version] 대화칸

이 례에서는 [v1_0]을 선택하여 [OK]단추를 찰칵하면 비교결과가 나타난다(그림 9-52).

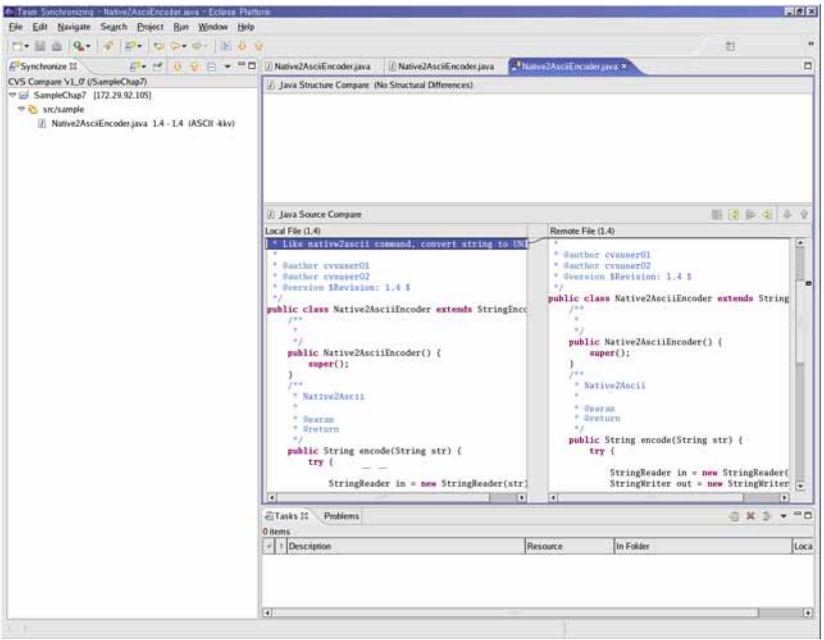


그림 9-52. 비교결과의 현시

아래에서 보게 되는 [보관고안의 서로 다른 2개 판본의 비교]에서 사용하기 위해 여기에서 현재의 SampleChap7에 판본[v1_1]을 붙여놓자. 즉 판본은 작업공간상의 프로젝트는 아니고 보관고안의 상태에 대하여 붙인것이다. 작업공간상에 보관하지 않은 변경이 있는 경우 보관하여야 하기때문에 판본을 붙일 필요가 있다(그림 9-53).

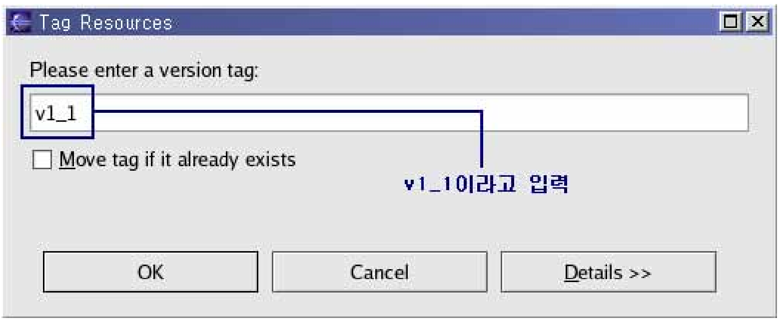


그림 9-53. 판본[v1_1]을 입력

— 보관고안의 서로 다른 2개 판본에서의 파일비교

보관고안에 여러개 판본이 존재하는 경우 [CVS Repository Exploring]로부터 그것들을 비교할수 있다. 여기서는 SampleChap7프로젝트의 개발성원 cvsuser01이 [v1_0]과 [v1_1]의 차이를 조사하여 내용비교를 진행한다.

① [CVS Repository Exploring]투시로 절환한 다음 [CVS Repository]보기의 [Version]을 찰칵하여 전개한다.

② 비교대상판본을 2개 선택한다. 이 례에서는 [v1_0]과 [v1_1]를 선택한다(그림 9-54).

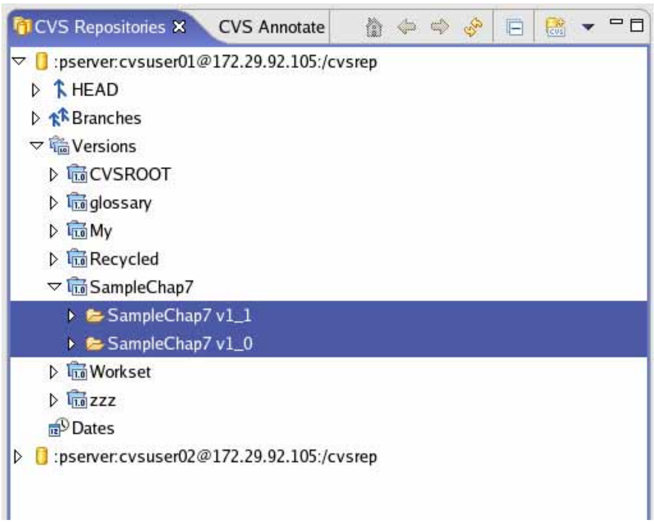


그림 9-54. 비교대상판본의 선택

③ 내부차림표로부터 [Compare]를 선택하면 비교결과가 현시된다.

[Structure Compare]에서 Native2AsciiEncoder.java를 두번찰칵한다(그림 9-55).

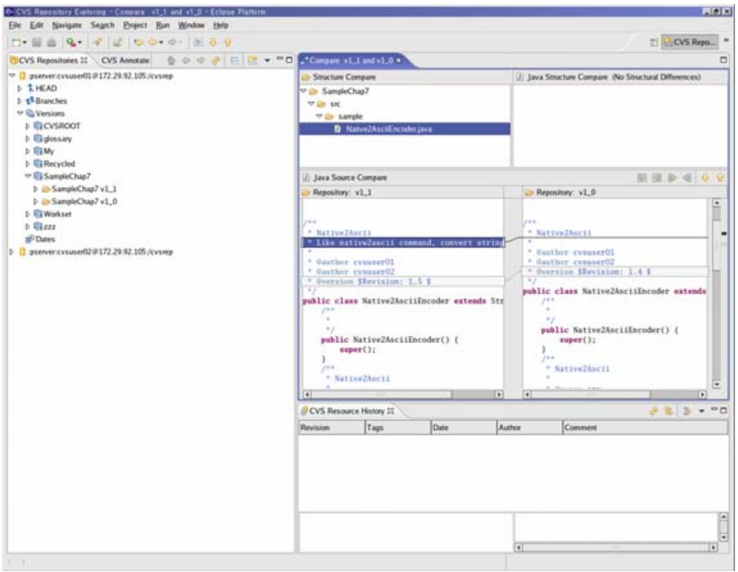


그림 9-55. 비교결과를 현시

9.5.4. 이전의 판본에로 되돌아가기

CVS에서 개발하는 경우 정확하게 동작하던 때의 판본으로 되돌아가야 할 경우가 있다. 이러한 경우 CVS에서는 과거의 판본내용을 작업공간에 올려쓰고 그것을 다시 보관고에 보관하는 조작을 할수 있다. 여기서는 SampleChap7프로젝트의 개발팀성원 cvsuser01이 프로젝트안의 내용을 v1_0때에로 되돌리려는 경우에 대하여 보기로 한다.

순서는 9.5.3.에서 본 [작업공간과의 비교]와 꼭같기때문에 되돌리려는 판본을 선택하는데 까지는 략한다.

① [Synchronize]보기의 [SampleChap7]을 오른쪽찰칵하고 내부차림표로부터 [Override or Update]을 선택한다.

② 작업공간의 올려쓰기에 대한 확인대화칸이 표시된다. [OK]단추를 찰칵하면 보관고의 [v1_0]의 내용이 작업공간에 복사된다(그림 9-56).

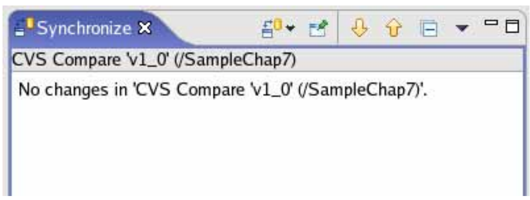


그림 9-56. v1_0의 내용과 일치시켰으므로 변경점이 없어진다

③ 보관을 진행한다.

제6절. 가지(Branch)

이 장의 지금까지의 례에서는 보관고안의 리력을 언제나 하나의 흐름처럼 취급해왔다. 파일의 교정이 1.1→1.2→1.3→1.4 이렇게 진행되고있는 이 《흐름》을 CVS의 용어로는 《**줄기**(trunk)》라고 한다.

개인 또는 여러명짜리 팀개발이라면 언제나 줄기만을 대상으로 개발을 진행하는것도 가능하다. 그러나 실제의 체계개발에 있어서는 이와 같이 단순하지 않다. 례들어 아래와 같은 상황에 대하여 생각해본다.

자신이 어떤 응용프로그람의 개발팀에 속해있다. 이전에 출하를 기다리던 판본1.0을 완성하여 구입자에게 넘겨주었다. 그 상태를 기록하기 위하여 보관고에는 판본 [v1_0]을 붙였다. 그후 오유보고가 1건, 기능추가요청이 1건 제기되였다. 오유수정에는 개발자 3명과 2주일의 시간을 투하하고 수정판 [v1_1] 로 하여 공개하도록 되였다. 기능추가는 좀 더 고려하여 5명으로 된 팀이 조직되여 1개월동안에 [판본 1.5]로서 공개판을 내놓도록 하였다.

그러면 어떻게 할것인가. 오유수정, 기능추가의 량팀쌍방이 줄기로부터 프로젝트를 꺼내서 각각 수정을 조금씩 하면서 보관하면 어떻게 될것인가. 이렇게 하면 때때로 다투기가 발생하고 작업에 지장을 줄수 있다. 례하면 쌍방에서 수정할 파일이 달라지고 다투기가 발생하지 않았어도 《기능추가의 도중상태》에 《오유수정의 도중상태》코드를 혼합하는것으로 된다. 이것은 절대로 허용될수 없는 일이다. 역시 오유수정팀, 기능추가팀의 쌍방에 한개의 최신안정판인 [v1_0]을 기초로 하였기때문에 각각 전용의 개발흐름이 요구된다. CVS의 《가지》기능을 사용하면 이것을 실현할수 있다. 이 경우 오유수정, 기능추가를 실현할수 있는 각각의 전용가지를 작성하고 그림 9-57과 같은 병행작업을 진행하면 된다.

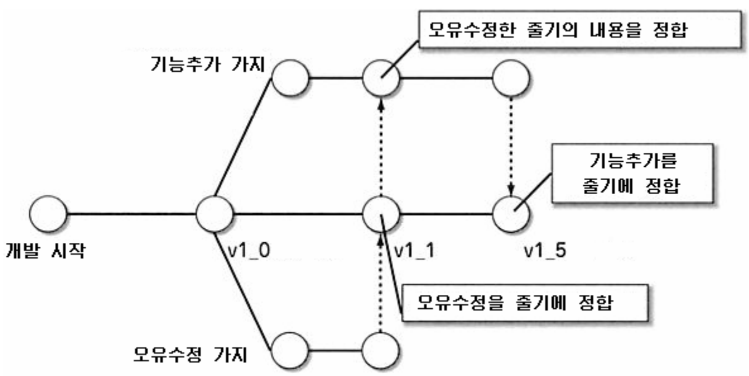


그림 9-57. 가지를 리용한 병행작업

여기서는 가지를 리용한 실례에 대하여 보기로 한다. 우에서 설명한 여러개 가지에 의한 병행개발은 약간 설명이 복잡해지기때문에 아래와 같은 간단한 경우에 대하여 취급하기로 한다.

SampleChap7개발팀에 있어서 Rot13Encoder.java의 개편을 진행한다. 변경은 [Branch01]에서 진행한다.

— 가지의 작성

① [Package Explorer]보기에서 프로젝트를 오른쪽찰칵하고 내부차림표로부터 [Team]→[Branch]를 선택한다.

② [Create a new CVS Branch]대화칸이 현시되는데 가지이름 《Branch01》을 입력한다.

[Version Name]은 가지를 작성하는것과 동시에 줄기상에 붙은 판본(태그)이다. 후에 [가지의 정합]작업을 진행할 때 요구된다(그림 9-58).

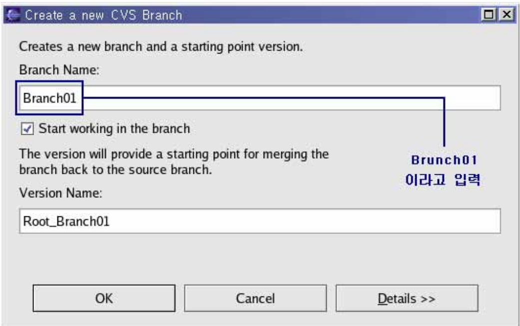


그림 9-58. 가지이름의 입력

[OK]단추를 찰칵한다. [Start working in the branch]칸을 선택하면 보관고안에 가지가 작성될뿐아니라 작업공간상의 프로젝트와 련관되기도 한다.

③ 프로젝트의 속성을 보면서 가지와 관련되였는가를 확인해본다(그림 9-59).

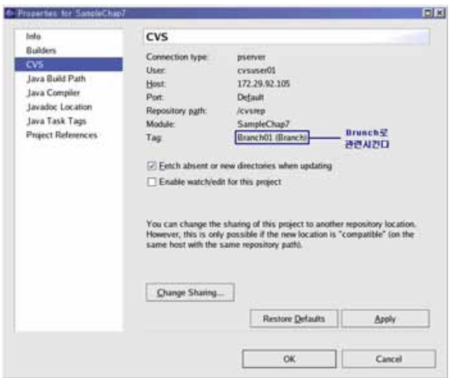


그림 9-59. 프로젝트와 가지의 련관

명확히 프로젝트가 [Branch01]에 관련된다.

이후의 변경은 줄기가 아니라 가지에 대응되여 보관하게 된다.

다음에 나오는 [가지의 정합]에서는 가지상에서 진행한 변경을 줄기에 정합하는 방법을 설명하는데 이것을 실현하기 위해서 가지상에서 몇가지 변경을 진행해본다. 여기에서는 [Rot13Encoder.java]의 클라스설명에 아래와 같은 변경을 가해본다.

```
/**
 * ROT13방식의 부호화기
 * ROT13이란 자모순의 매 문자를 13문자씩 편차시킨것
 * 단순히 읽기 어려운 알고리듬
 *
 * @author cvsuser01
 * @version $Revision$
 */
```

— 가지의 정합

가지상에서 진행한 변경을 줄기(또는 다른 가지)에 적용하는것을 가지를 정합한다고 한다. 여기에서는 SampleChap7프로젝트의 개발팀성원 cvsuser01이 가지 [Branch01]의 내용을 줄기(HEAD)에 정합하는 방법에 대하여 보기로 한다(그림 9-60).

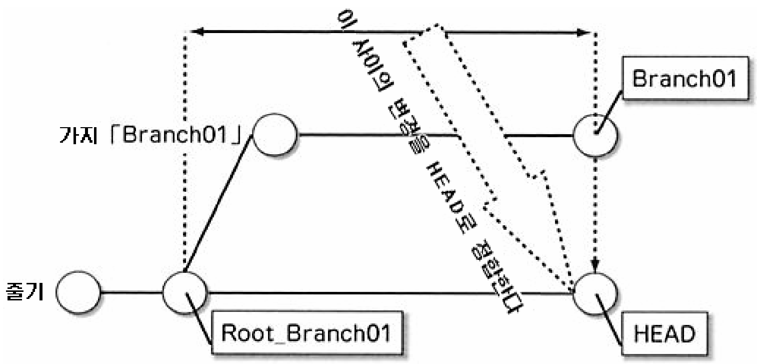


그림 9-60. 정합작업

Eclipse에서의 가지정합은 몇개의 가지를 현재의 작업공간에 정합하여 보관하는 방법이다.

① 작업공간은 정합하기 전(이 례의 경우 HEAD)으로 절환한다.

[Package Explorer]보기에서 프로젝트를 오른쪽찰칵하여 현시되는 내부차림표에서 [Replace with]→[Another Branch or Version]을 선택한다.

② [Replace with Branch or Version]대화칸이 나타난다.

[HEAD]를 선택하여 [OK]단추를 찰칵한다(그림 9-61).



그림 9-61. 절환하려는 가지를 선택

③ 정합을 시작한다.

[Package Explorer]보기에서 [SampleChap7]을 오른쪽찰칵하고 [Team]→[Merge]을 선택한다. 그러면 [Select the merge start point]대화칸이 현시되는데 가지를 만든 경우에 줄기에 붙은 판본인 [Root_Branch01]을 선택하고 [Next]단추를 찰칵한다(그림 9-62).

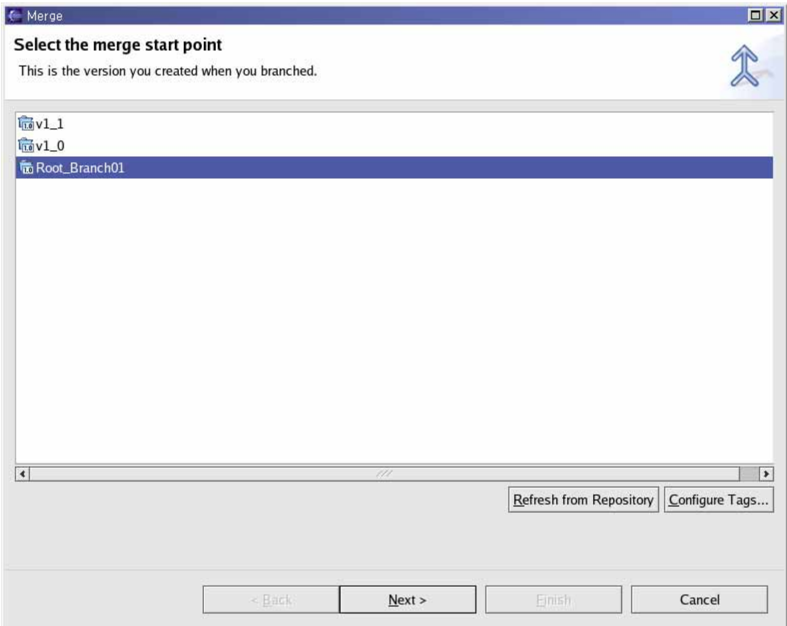


그림 9-62. 정합의 시작점을 선택

④ 다음에 정합하려는 가지를 선택한다. 이 실례에서는 [Branch01]을 선택한다(그림 9-63).

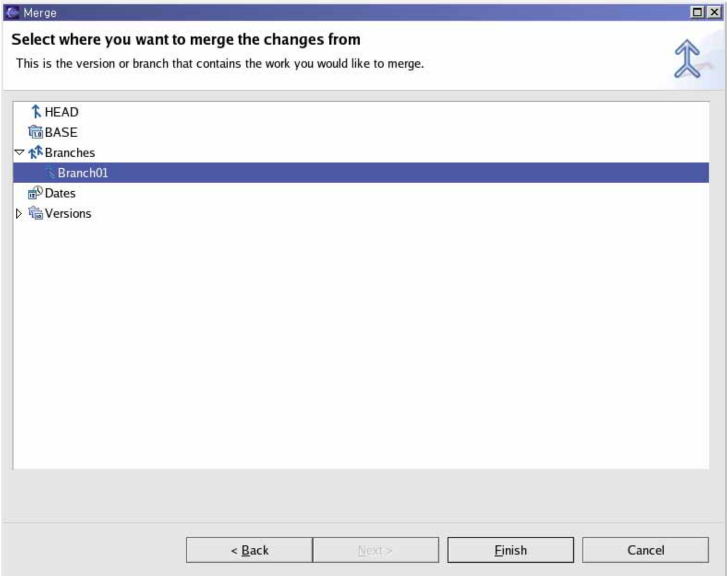


그림 9-63. 정합한 가지의 선택

[Finish]단추를 찰칵하면 정합원(Branch01)과 정합선(HEAD)의 비교작업이 진행되고 결과가 나타난다. [Synchronize]보기의 [ROT13Encoder.java]를 두번찰칵한다(그림 9-64).

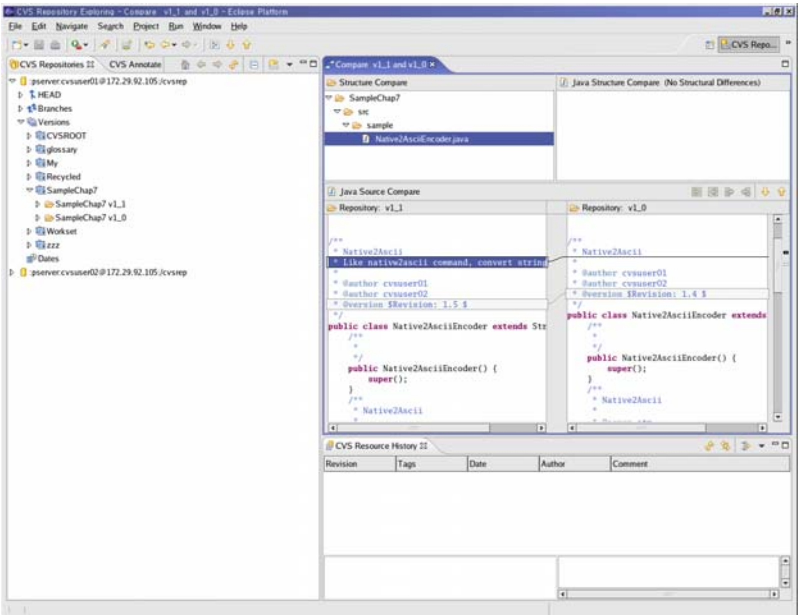


그림 9-64. 비교결과

⑤ [Synchronize]보기에서 [SampleChap7]을 오른쪽찰칵하여 내부차림표로부터 [Refresh]을 선택하면 가지의 내용이 작업공간에서 정합된다. [Synchronize]보기가 그림 9-65와 같이 되면 정합은 끝난다.

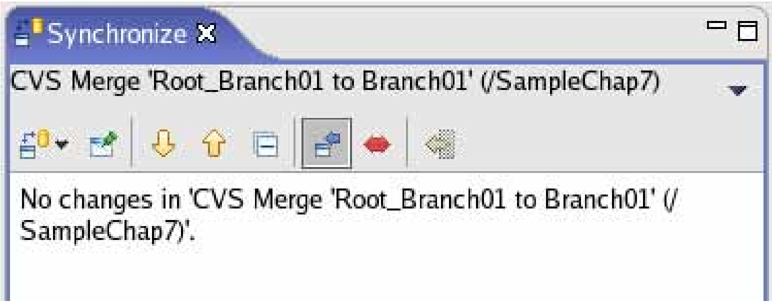


그림 9-65. 정합의 완료

⑥ 반드시 작업공간에 변경이 있기때문에 보관을 진행한다.

제10장. Web응용프로그람의 개발

제1절. 개요

10.1.1. 끼워넣기(plugin)를 리용한 Web응용프로그람봉사기와의 련동

Eclipse는 끼워넣기에 의해 기능을 확장할수 있으며 이미 많은 끼워넣기가 제공되여있다.

이 장에서는 Web응용프로그람봉사기와 련동을 진행하는 끼워넣기를 도입하고 Web응용프로그람의 개발, 실행 및 오유수정을 진행하는 방법에 대하여 소개한다. Web응용프로그람봉사기로서 Jakarta프로젝트의 Tomcat를 리용한다. 또한 Tomcat와 련동을 진행하는 끼워넣기로서 Lomboz를 리용한다.

10.1.2. Lomboz

Lomboz란 Eclipse상에서 J2EE응용프로그람개발을 지원하는 끼워넣기이다. 응용프로그람의 개발, 배치, 응용프로그람봉사기의 련동, 시험, 오유수정, 정지라고 하는 일련의 개발공정을 Eclipse상에서 진행할수 있다.

지원하는 응용프로그람봉사기로서 Tomcat, JBoss, WebLogic 등을 들수 있다. JSP의 코드형식에 따르는 색표시, 코드편집방조기 기능을 갖춘 JSP편집기도 제공해놓고 EJB개발지원기능도 갖추고있다.

o Lomboz의 주요한 기능

- 응용프로그람봉사기(Tomcat, JBoss, WebLogic 등)의 기동, 정지
- Web응용프로그람(HTML, Servlet, JSP)개발지원
- J2EE Web응용프로그람압축형식(Ear/War), EJB압축형식(Jar)작성과 배치(Ant를 리용)
- 응용프로그람봉사기와 련동하는 봉사기측코드(JSP, Servlet, EJB)의 오유수정
- JSP코드형식에 따르는 색표시, 코드편집방조기, 문법검사
- EJB 1.1, 2.0(Session, Entity, Message Driven Bean)의 개발지원(XDoclet를 리용)
- EJB시험말단기의 생성
- Web봉사기말단기의 생성(Applet Axis를 리용)

제2절. 설치와 설정

10.2.1. Tomcat의 설치와 설정

1. Tomcat의 입수

이 장에서는 Jakarta Tomcat 5.0.28을 전제로 하여 서술한다.

여기서는 이 파일을 리용하여 진행하는 설치를 보기로 한다. Tomcat의 최신판은 《http://jakarta.apache.org/tomcat/index.html》로부터 내리적재할수 있다.

2. 설치와 설정

Jakarta tomcat 5.0.28.zip를 리용하여 Tomcat의 설치를 진행한다.

Jakarta tomcat 5.0.28.zip를 적당한 장소에 풀면 Jakarta tomcat 5.0.28이라는 등록부가 생긴다. 풀기가 끝나면 등록부이름을 Tomcat로 변경시켜준다. 이렇게 설치를 완료한다. 설치후 Tomcat가 참조하는 환경변수를 설정한다. 《CATALINA_HOME = /kut/Tomcat》라고 설정한다.

3. 기동의 확인

다음으로 Tomcat의 기동에 대하여 확인한다.

열람기상에서 우에서 서술한 CATALINA_HOME에서 설정한 등록부아래의 bin등록부안에 있는 startup.sh를 두번 찰칵하여 기동한다.

조종탁으로부터 아래와 같이 실행한다.

sh startup.sh 또는 .startup

여기에서는 JDK1.6.0과 환경변수 JAVA_HOME이 이미 설정되였있다는것을 전제로 하고있다. 설정되여있지 않는 경우 CATALINA_HOME을 설정한것과 같은 방법으로 JAVA_HOME으로서 《/usr/java/jdk1.6.0》을 설정한다. 그림 10-1과 같은 화면이 현시되고 Tomcat가 기동한다.

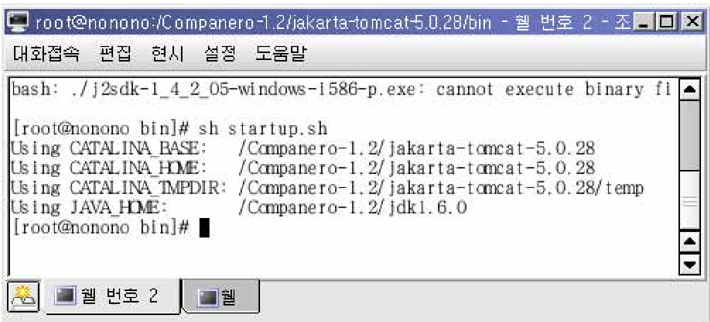


그림 10-1. Tomcat의 기동화면

열람기를 기동하고 《http://localhost:8080》으로 접근조종한다. 그림 10-2와 같은 화면이 현시되면 성공한것이다.

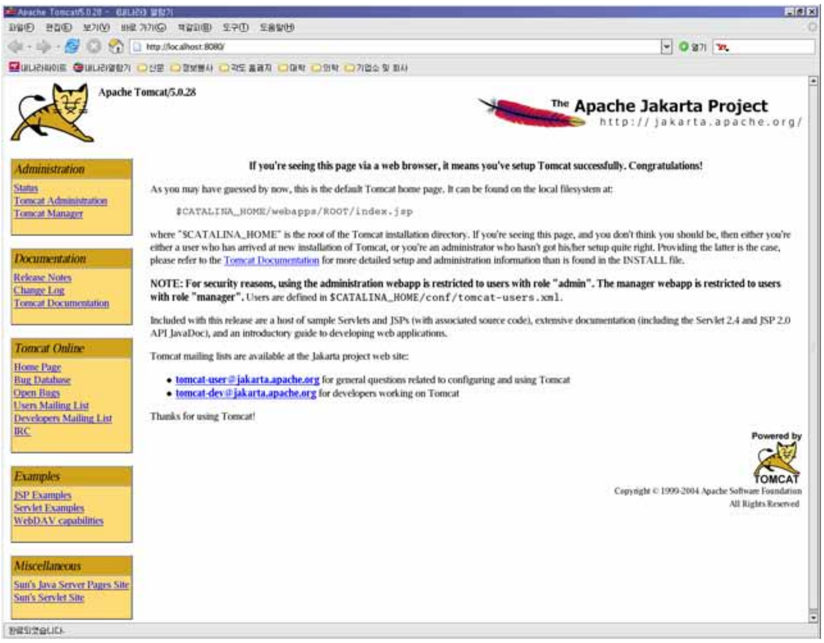


그림 10-2. Tomcat의 화면

이렇게 기동에 대한 확인을 한다.

Tomcat를 정지시킨다. 파일열람기로부터 《/kut/Tomcat/bin》등록부에 있는 shutdown.sh를 두번 찰칵하면 정지된다. 조종탁으로부터 아래와 같이 실행할수 있다.

sh shutdown.sh 또는 .shutdown

10.2.2. Lomboz의 설치와 설정

1. Lomboz의 입수

여기서는 Lomboz 3.0.1을 전제로 하여 서술하였다. 그리고 Lomboz의 최신판은 《http://www.objectlearn.com/》로부터 내리적재 할수 있다.

Lomboz 3.0을 리용할 때에는 eclipse EMF실행시간용 2.0이상의 판본도 설치할 필요가 있다. EMF실행시간의 최신판은 《http://download.eclipse.org/tools/emf/scripts/downloads.php》로부터 입수할수 있다.

2. 설치

Lomboz.301.zip를 리용하여 Lomboz의 설치를 진행한다. 여기에서 언급하는 zip형식의 파일을 적당한 장소에 풀어놓으면 plugins등록부가 생긴다. 풀기가 끝나면

plugins등록부안의 전체파일을 /kut/eclipse의 plugins등록부안에 배치한다. 이상으로 설치는 완료된다.

Lomboz 3.0.1과 Tomcat 5.0.28을 조합시켜 리용하는데서 주의할 점

Lomboz 3.0.1과 Tomcat 5.0.28을 조합시키면 Eclipse에서는 Tomcat가 기동할 때 례외(java.lang.NullPointerException)가 발생하고 정확히 기동하지 않는다. 이것은 Tomcat에 부속되여있는 commons-logging-api.jar의 판본이 Tomcat 5.0.27에서 1.0.3으로부터 1.0.4로 변경되였기때문이다. 이 문제는 아래에 서술한 두가지 방법중의 하나를 적용하는것으로 풀수 있다.

1. commons-logging-api.jar의 판본을 1.0.3으로 바꾼다.
   commons-logging-api.jar(판본1.0.3)을 \Companero-1.2\jakarta-tomcat-5.0.28\bin에 복사한다.
2. Tomcat기동할 때의 인수 (-Djava.endorsed.dirs)로부터 《${serverRootDirectory}/bin》을 소거한다.
   Lomboz의 Tomcat5.0.x용 봉사기설정파일 (\KUT\eclipse\plugins\com.objectlearn.jdt.j2ee_3.0.1\server\tomcat50x.server)을 편집기로 열고 인수 (-Djava.endorsed.dirs)로부터 《${serverRootDirectory}/bin》을 소거한다.(Tomcat의 기동과 완료의 설정부분을 설정할 필요가 있다.)

3. 설정

Eclipse를 기동하고 아래와 같이 설정한다.

[Java]투시가 열려져있지 않는 경우 차림표띠로부터 [Window]→[Open Perspective]→[Java]를 선택하고 [Java]투시를 연다. 다음 차림표띠로부터 [Window]→[Customize Perspective]를 선택하면 현시되는 대화칸에서 [Shortcuts]태브를 선택한다. [Submenus]에서 [New]을 선택하고 [Shortcuts Categories]에서 [Java]→[Lomboz J2EE Wizards]를 선택한다(그림 10-3). 꼭같이 [Submenus]에서 [Show View]를 선택하고 [Shortcuts Categories]에서 [Lomboz J2EE]를 선택한다.

다음에 [Commands]태브를 선택하고 [Available command groups]에서 [Lomboz Actions]를 선택한 다음 [OK]단추를 찰칵한다.

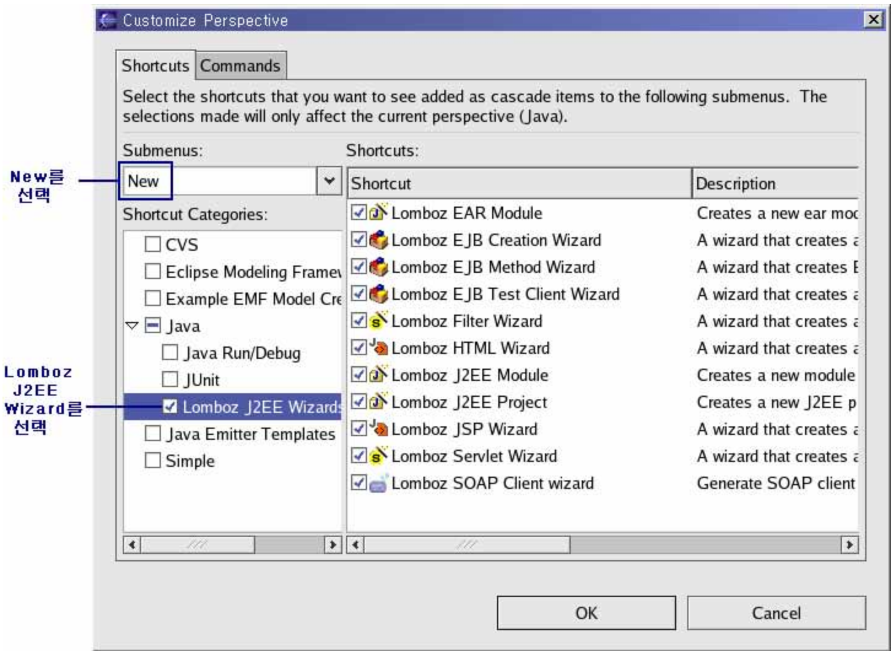


그림 10-3. 투시의 전용화대화칸

그림 10-4와 같이 차림표띠로부터 [File]→[New]를 선택하면 현시되는 부분차림표에서 Lomboz관련의 항목이 등록되여있는가를 확인한다.

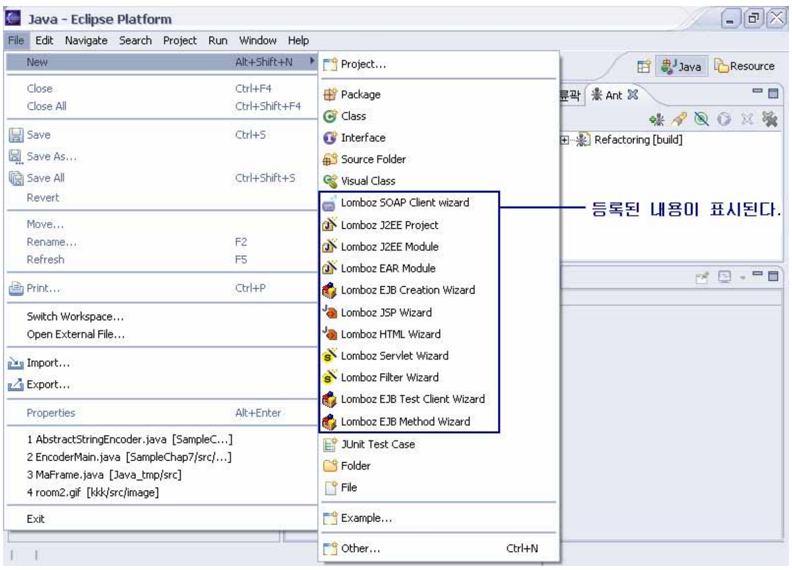


그림 10-4. 등록내용의 확인

다음 차림표띠로부터 [Window]→[Preferences]을 선택하고 [Preferences]대화칸을 연다. 대화칸왼쪽의 [Workbench] → [Label Decorations]을 선택한 다음 [Avaliable label decorations]의 [Lomboz J2EE Decorators]를 선택한다(그림 10-5).

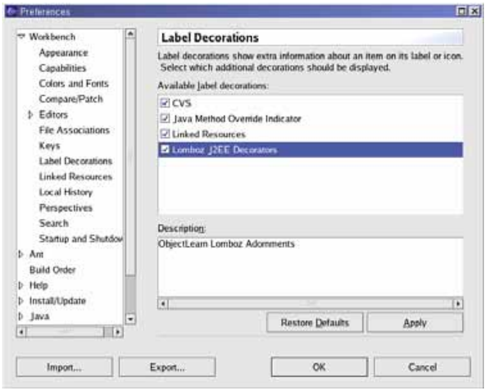


그림 10-5. 표시장식설정대화칸

같은 방법으로 [Preferences]대화칸에서 [Java]→[Buildpath]를 선택하고 프로젝트가 기정으로 사용하는 건립경로를 지정한다. 여기서는 [Folders]를 선택한다. [Source folder name]이 《src》, [Output folder name]이 《bin》으로 되여있는가를 확인한다(그림 10-6).

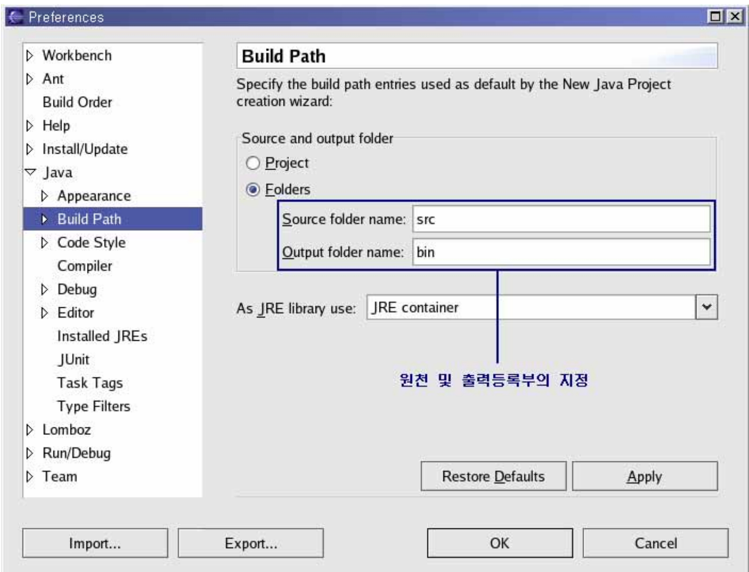


그림 10-6. 건립경로설정대화칸

대화칸왼쪽의 [Lomboz]를 선택하고 [JDK Tools.jar]의 경로를 설정한다. 여기서는 JDK1.6.0이 《/usr/java/jdk1.6.0》 또는 《 /Companero-1.2/jdk1.6.0》에 설치되였기때문에 《/usr/java/jdk1.6.0/lib/tools.jar》 또는 《/Companero-1.2/jdk1.6.0/lib/tools.jar》를 설정한다(그림 10-7).

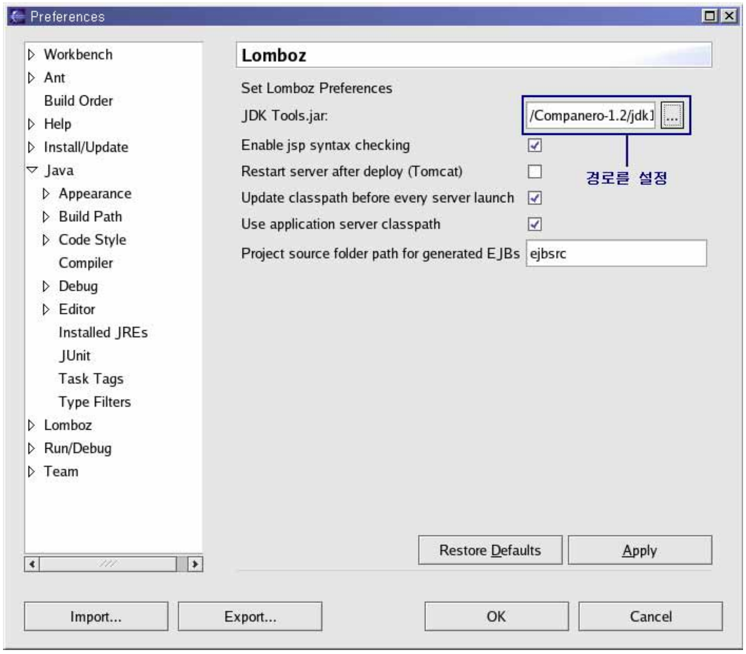


그림 10-7. Lomboz설정대화칸

[Lomboz]→[Server Definitions]를 선택하여 응용프로그람봉사기에 대한 설정을 진행한다. 이 장에서는 Tomcat 5.0.28을 사용하기때문에 [Server Types]에 《Apache Tomcat v5.0.x》를 선택하고 [Properties]태브에서 [Application Server Directory] 및 [Classpath Variable]에 《/Companero-1.2/jakarta-tomcat-5.0.28》을 설정한 다음 [Apply]단추를 찰칵한다(그림 10-8).

[Java]→[Build Path]→[Classpath Variables]를 선택하고 그림 10-9와 같이 Tomcat에 대한 클라스경로변수가 설정되여있는가를 확인한다. 다음 [OK]단추를 찰칵하여 [Preferences]대화칸을 닫는다. 이상으로 Lomboz의 설정은 끝난다.

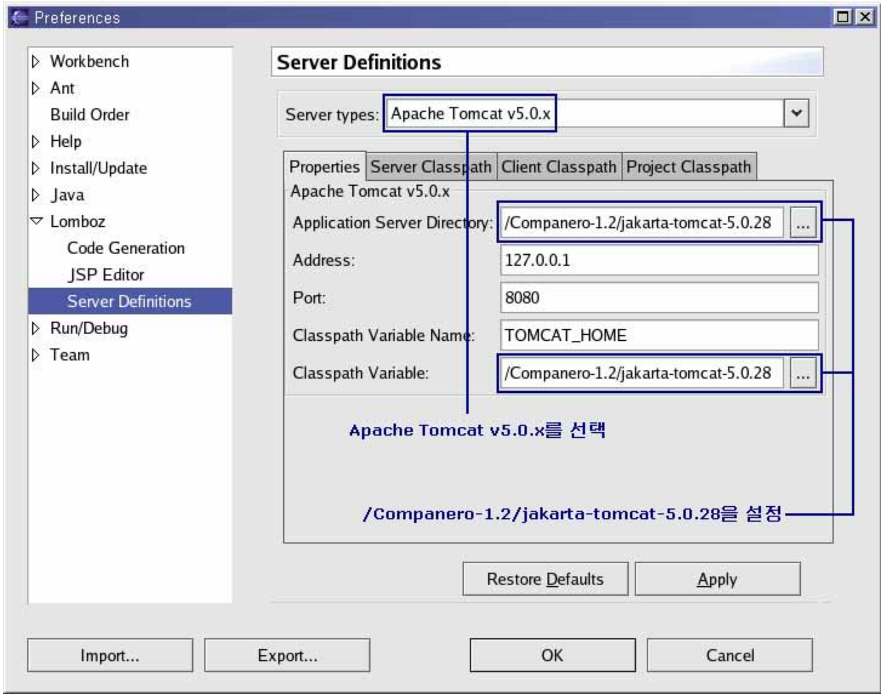


그림 10-8. 봉사기설정대화칸

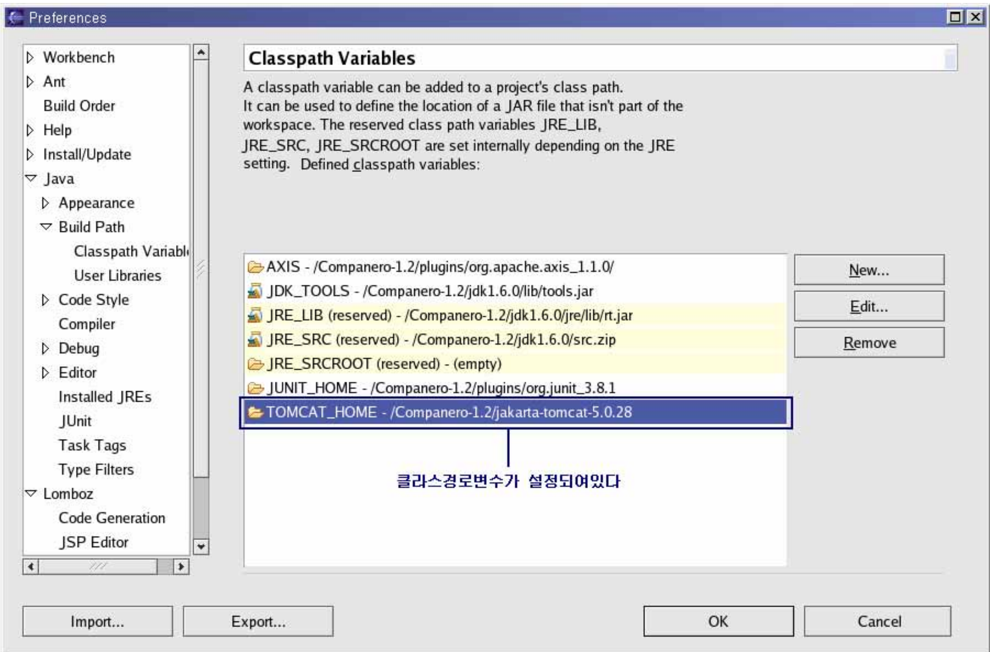


그림 10-9. 클라스경로변수설정대화칸

제3절. Web응용프로그람작성과 실행

10.3.1. Web응용프로그람작성

종업원검색에 대한 간단한 응용프로그람을 작성함으로서 Eclipse상에서의 Web응용프로그람개발에 대하여 학습한다. 여기에서 작성하는 종업원검색응용프로그람은 JSP, Servlet 및 검색엔진을 가진 Java클라스를 탑제하는것으로 처리를 진행한다(그림 10-10).

이 처리모델을 MVC모델이라고 부른다. MVC모델은 응용프로그람을 업무론리를 가지는 Model, 표시론리를 가지는 View, 그리고 Model과 View를 조종하는 Controller의 3가지로 구성되는 형태로서 Model과 View를 명확히 분리시키고 응용프로그람설계의 유연성, 재리용성을 향상시킬수 있다. 즉 JSP, Servlet를 기초로 한 Web응용프로그람에서는 JSP가 View, Servlet가 Controller의 역할을 담당한다.

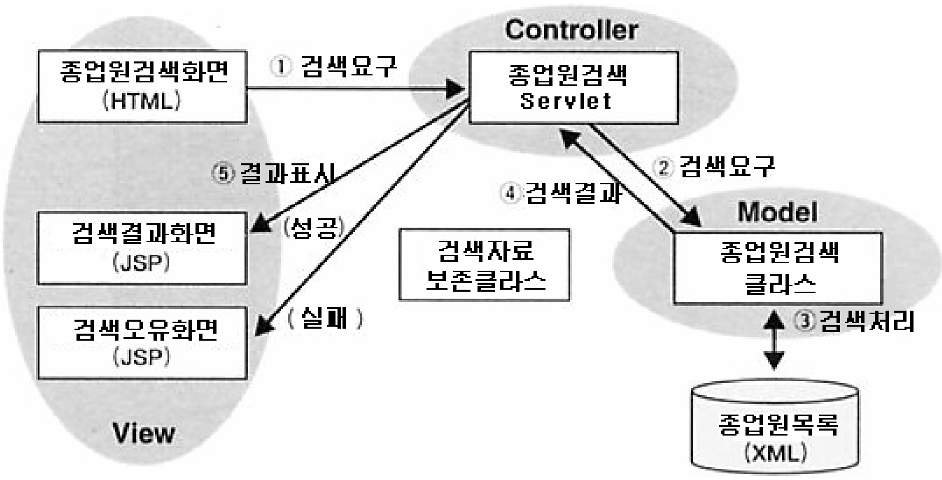


그림 10-10. 종업원검색응용프로그람의 처리방식

표 10-1에 종업원검색응용프로그람의 각 구성요소의 개요를 그림 10-11에는 종업원검색응용프로그람의 화면이행에 대하여 보여주었다.

표 10-1. 종업원검색응용프로그람의 구성요소들

명 칭	형 식	개 요
종업원검색화면	HTML	종업원검색의 열쇠어로 되는 종업원번호의 입력폼을 가진 화면이다. 종업원번호를 입력하여 종업원검색 Servlet에 검색요구를 준다.
종업원검색 Servlet	Servlet	종업원검색화면에서 보낸 종업원번호의 값을 읽어들이고 종업원검색클라스에 검색을 의뢰한다. 또한 검색결과에 따라 종업원검색결과화면 또는 종업원검색오유화면으로 결과를 보낸다.
종업원검색클라스	Java클라스	넘겨받은 종업원번호를 리용하여 종업원목록(XML)으로부터 종업원검색을 진행한다.
검색자료클라스	Java클라스	종업원검색 Servlet, 종업원검색클라스, 종업원검색결과 및 종업원검색오유화면에로 보내는 자료포함기
종업원검색결과 화면	JSP	종업원검색클라스로부터 자료를 받아 종업원정보를 현시한다.
종업원검색오유 화면	JSP	종업원검색클라스로부터 오유를 받아 오유정보를 현시한다.



그림 10-11. 종업원검색응용프로그람의 화면이행

10.3.2. 프로젝트의 작성

먼저 종업원검색응용프로그람개발을 위한 프로젝트를 작성한다.

Eclipse상의 차림표띠로부터 [File]→[New]→[Lomboz J2EE Project]를 선택하고 프로젝트작성조수대화칸을 연다(그림 10-12).

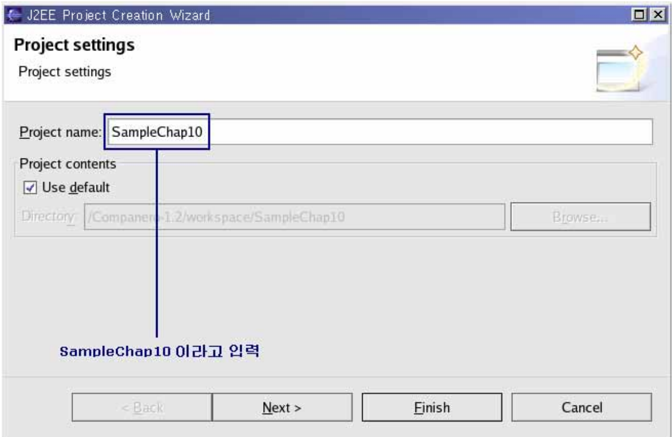


그림 10-12. 프로젝트작성조수대화칸

[Project name]에 《SampleChap10》을 입력하고 [Project contents]는 기정대로 설정한 다음 [Next]를 찰칵한다. 그러면 [Java Settings]대화칸이 현시된다. [Source]태브에서 [Source folder on build path]에는 《SampleChap10/src》등록부가, [Default output folder]에는 《SampleChap10/bin》등록부가 설정되여있는가를 확인한다(그림 10-13). [Next]를 찰칵한다.

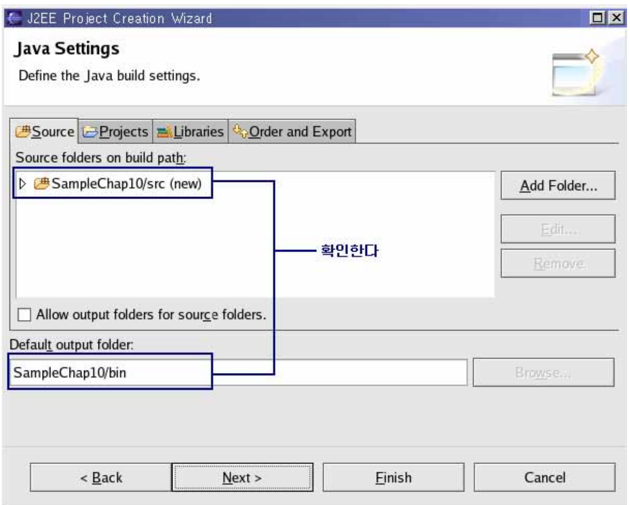


그림 10-13. Java설정대화칸

[Create J2EE Module]대화칸의 [Web Modules]태브에서 [Add]단추를 리용하여 모듈이름을 《SimpleWeb》으로 등록한다(그림 10-14).

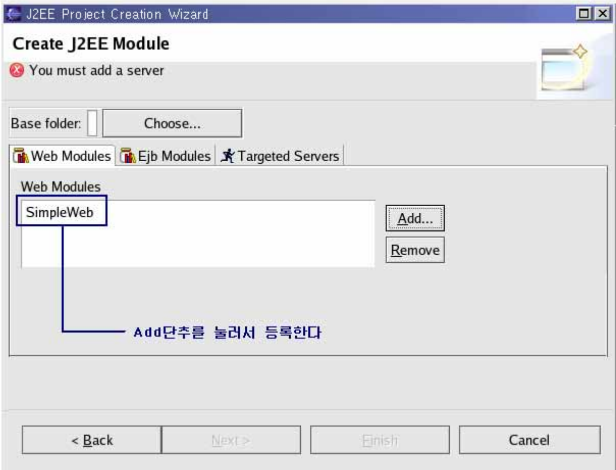


그림 10-14. J2EE모듈작성대화칸

[Targeted Servers]태브를 선택하여 봉사기의 설정확인을 진행한다. 여기서는 Tomcat 5.0.28을 리용하므로 [Type]에서 《Apache Tomcat v5.0.x》를 선택하고 [Add]를 찰칵한다. [Targeted Servers]로서 Tomcat가 등록되기때문에 [Finish]를 찰칵한다(그림 10-15).

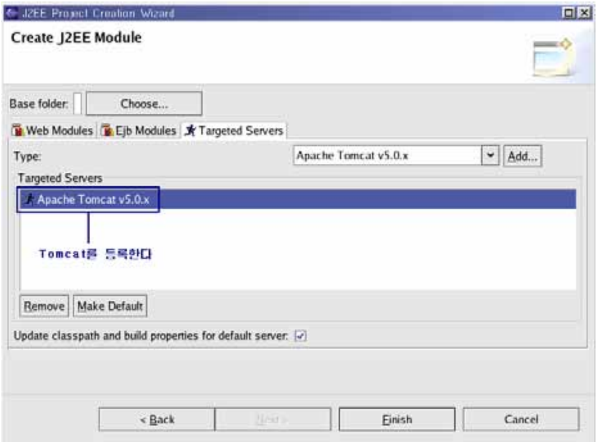


그림 10-15. Targeted Servers태브를 눌렀을 때 현시되는 대화칸

이것으로서 프로젝트의 작성은 완료된다. [Package Explorer]보기에서 SampleChap10 프로젝트가 등록되였는가를 확인한다(그림 10-16).

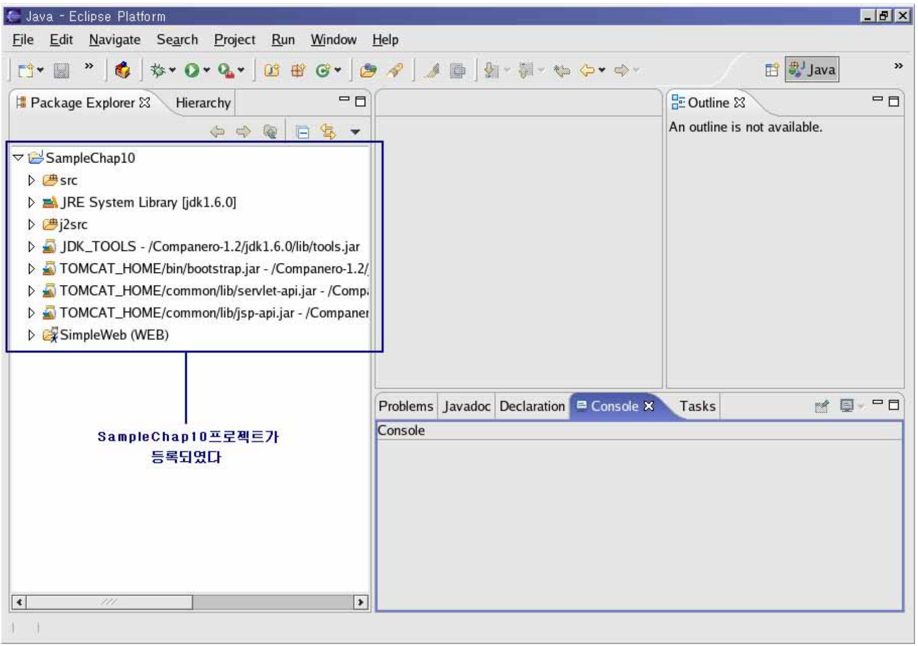


그림 10-16. 작성된 SampleChap10프로젝트

10.3.3. 종업원검색화면(HTML)의 작성

프로젝트가 작성되였으므로 종업원검색응용프로그람의 개발을 시작해본다. 처음에 종업원검색용화면(index.html)을 작성한다. [Navigator]보기 또는 [Package Explorer]보기상에서 [SampleChap10]→[SimpleWeb]를 오른쪽찰칵하고 [New]→[Lomboz HTML Wizard]를 선택하여 HTML작성대화칸을 연다. [File name]에《index.html》이라고 입력하고 [Finish]를 찰칵한다(그림 10-17).

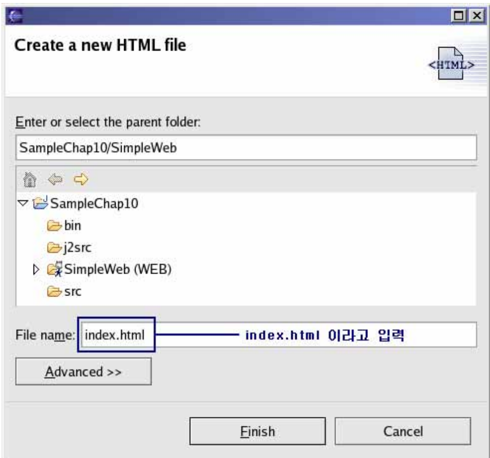


그림 10-17. HTML작성대화칸

[Navigator]보기가 현시되여있지 않는 경우에는 차림표띠로부터 [Window]→[Show View]→[Navigator]를 선택하면 현시된다. Index.html이 작성되면 편집기가 현시되는데 여기에 목록 10-1과 같은 코드를 입력한다. 여기서 굵은 글자로 되여있는 부분은 추가 또는 수정한 코드이다. [종업원검색화면]에서는 [검색결과]단추를 찰칵하면 종업원번호(memberId)를 파라메터로 하여 종업원검색Servlet(/SimpleWeb/MemberSearchServlet)가 기동하도록 설정한다.

목록 10-1. 종업원검색화면 (index.html)

```
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<html>
<head>
<title>종업원검색</title>
</head>
<body bgcolor="#FFFFFF">
<div align="center">
<h3>검색하려는 종업원번호를 입력한다.</h3>
<form name="MemberSearchForm" method="post"
        action="/SimpleWeb/MemberSearchServlet">
<table width="300" cellpadding=10 align="center">
  <tr>
    <td width="40%" bgcolor="lightblue"><div align="center">종업원번호
          </div></td>
    <td  width="60%"  bgcolor="palegoldenrod"><input  type="text"
          name="memberId"></td>
  </tr>
</table>
<br>
<input type="submit" name="submit" value="검색진행">
</form>
</div>
</body>
</html>
```

작성한 index.html파일을 열람기상에서 실행하면 그림 10-18과 같이 현시된다.

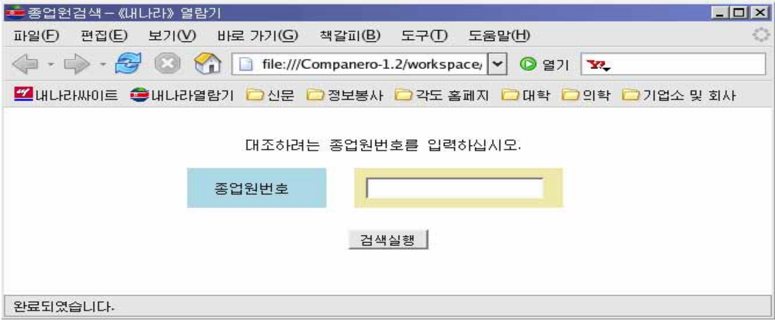


그림 10-18. 종업원검색화면

10.3.4. 종업원검색Servlet의 작성

종업원검색Servlet(MemberSearchServlet.java)를 작성한다. [Navigator]보기상에서 [SampleChap10]→[src]를 오른쪽찰칵하고 [New(새로 작성)]→[Lomboz Servlet Wizard]를 선택하여 대화칸을 현시한다. 여기에서 [Package]에 《sample.web》를, [Name]에는 《MemberSearchServlet》를 입력한다. [Modifiers]에서 [public]를 선택하고 [Superclass]에 《java.servlet.http.HttpServlet》라고 설정하며 [Which method stubs would you like to create?]에서 [doPost()]를 선택한 다음 [Next]를 찰칵한다(그림 10-19).

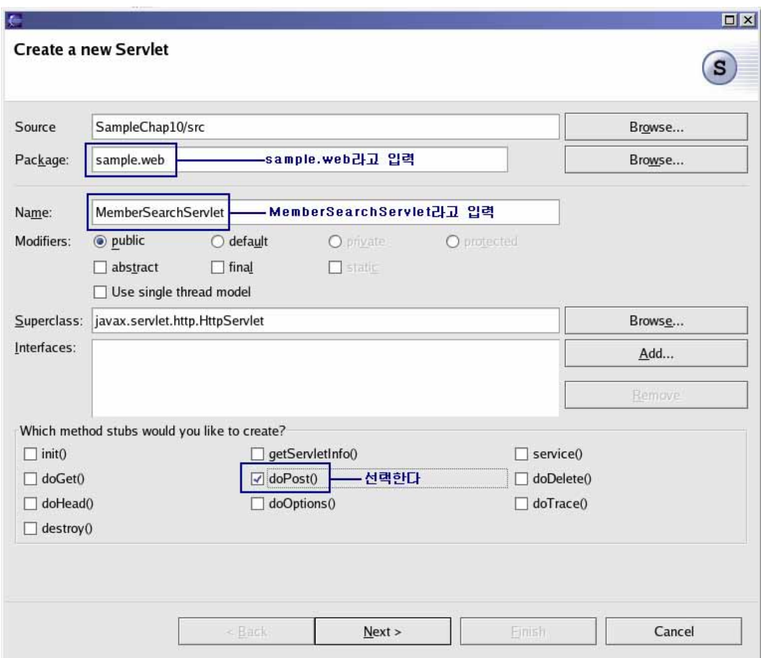


그림 10-19. Servlet작성대화칸

그러면 배치서술자설정대화칸이 현시되는데 여기서 [Web module]에는 《SimpleWeb》, [Servlet Name]에 《MemberSearchServlet》, [Mapping URL]에 《/MemberSearchServlet/*》를 설정한 다음 [Finish]를 찰칵한다(그림 10-20).

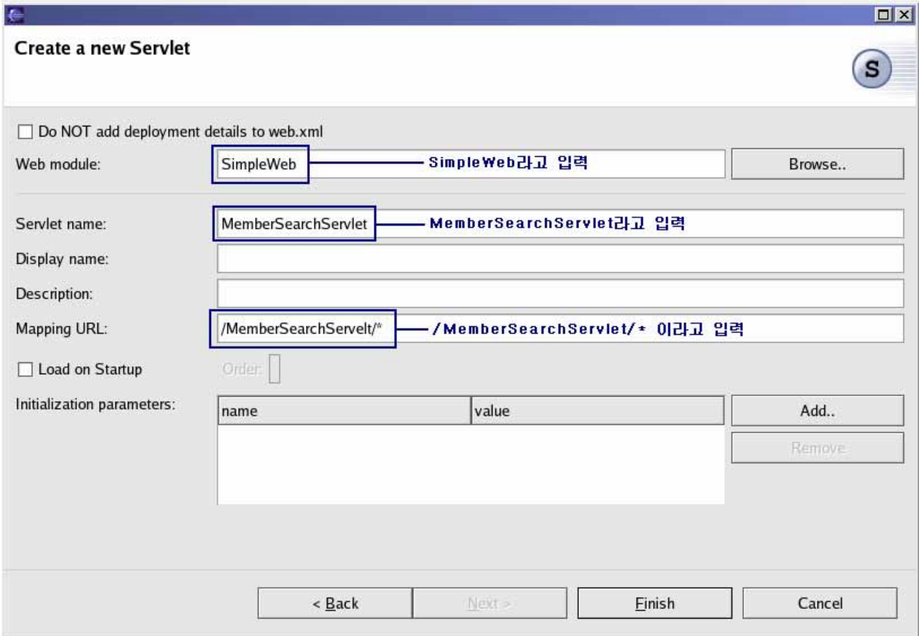


그림 10-20. 배치서술대화칸

우와 같은 설정을 진행하면 http://localhost:8080/SimpleWeb/MemberSearchServlet로 종업원검색Servlet가 기동되도록 배치서술자(web.xml)의 설정이 자동적으로 추가된다.(상세한것은 후에 서술한다.)

MemberSearchServlet.java가 작성되여 편집기에 현시되는데 여기에 목록 10-2와 같은 코드를 입력한다.(굵은 문자로 되여있는 부분은 추가 또는 수정한 코드이다.) 종업원검색servlet에서는 먼저 요청(request)으로부터 얻어진 종업원번호(memberId)를 가지고 종업원검색클라스(MemberSearch)에 검색을 의뢰한다. 검색이 성공(값 1을 귀환)이면 종업원검색결과화면(/memberSearchOK.jsp)이 현시되고 검색이 실패(검색결과가 없거나 례외가 발생)하면 종업원검색오유화면(/memberSearchNG.jsp)이 현시된다. 이 시점에서는 [종업원검색클라스]작성이 완료되지 않았으므로 번역시 오유가 발생한다. 다음과 같이 작성하면 된다.

목록 10-2. 종업원검색Servlet(MemberSearchServlet.java)

```
/*
 * Created on May 25, 2005
 *
 * To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
package simple.web;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @author Administrator
 *
 * To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
public class MemberSearchServlet extends HttpServlet {

    private static final String ngPage = "/memberSearchNG.jsp";
    private static final String okPage = "/memberSearchOK.jsp";

    protected void doPost(HttpServletRequest request,
                HttpServletResponse response)
                throws ServletException, IOException {

            String resultPage;
            MemberDataHolder mdh = new MemberDataHolder();
            mdh.setId(request.getParameter("memberId"));


            if(MemberSearch.search(mdh) == 1) {
                    resultPage = okPage;
            }
            else {
                    resultPage = ngPage;
            }

            request.setAttribute("mdh", mdh);
            javax.servlet.RequestDispatcher dispatch =
                    request.getRequestDispatcher(resultPage);
            dispatch.forward(request, response);
    }
}
```

다음에 배치서술자(web.xml)의 확인 및 편집을 진행한다. [Navigator]보기상에서 [SampleChap10]→[SimpleWeb]→[WEB-INF]→[web.xml]을 선택하여 web.xml파일을 현시한다. 이 파일에 http://localhost:8080/SimpleWeb/MemberSearchServlet를 실행하여 종업원검색Servlet를 기동하기 위한 아래와 같은 코드가 있는가를 확인해본다.

```
<servlet>
   <servlet-name>MemberSearchServlet</servlet-name>
   <servlet-class>simple.web.MemberSearchServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>MemberSearchServlet</servlet-name>
   <url-pattern>/MemberSearchServlet/*</url-pattern>
</servlet-mapping>
```

즉 이 web.xml파일에는 welcome-file-list 및 error-page가 더 정의되여있다.(Lomboz프로젝트의 작성시에 자동적으로 정의된다.) 이 책의 종업원검색응용프로그람에서는 리용할수 없기때문에 소거한다.

목록 10-3은 편집후의 web.xml파일이다.

목록 10-3. 배치서술자(web.xml)

```
<?xml version="1.0" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
         Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">
<!-- Copyright (c) 2002 by ObjectLearn. All Rights Reserved. -->
<web-app>
    <!-- Remove the comments below to define a servlet.      -->
    <!--  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
    <!--   <servlet>                                          -->
    <!--     <servlet-name>MyServlet</servlet-name>            -->
    <!--     <servlet-class>examples.MyServlet</servlet-class> -->
    <!--     <init-param>                                     -->
    <!--       <param-name>myparam</param-name>               -->
    <!--       <param-value>12345</param-value>               -->
    <!--   </init-param>                                      -->
    <!--  </servlet>                                          -->
    <!--                                                      -->
    <!--                                                      -->
    <!--  <servlet-mapping>                                   -->
    <!--     <servlet-name>MyServlet</servlet-name>           -->
    <!--     <url-pattern>/mine/*</url-pattern>               -->
    <!--  </servlet-mapping>                                  -->
    <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++   -->
```

```
    <servlet>
       <servlet-name>MemberSearchServlet</servlet-name>
       <servlet-class>simple.web.MemberSearchServlet</servlet
      -class>
    </servlet>
    <servlet-mapping>
       <servlet-name>MemberSearchServlet</servlet-name>
       <url-pattern>/MemberSearchServlet/*</url-pattern>
    </servlet-mapping>

 </web-app>
```

10.3.5. 검색자료보존용 클라스의 작성

검색자료보존클라스(MemberDataHoder.java)를 작성한다.

[Navigator]보기에서 [SampleChap10]→[src]를 오른쪽찰칵하고 [New]→[Class]를 선택하면 Java클라스작성조수대화칸이 현시된다. 여기에서 [Package]에 《sample.web》를 입력하고 [Name]에는 《MemberDataHoder》를 입력한다. 그리고 [Modifiers:]에서 [public]를 선택하고 [Superclass]에는 《java.lang.Object》를 설정하며 [Which method stubs would you like to create?]에서는 모든 검사칸들을 선택한 다음 [Finish]를 찰칵한다(그림 10-21).

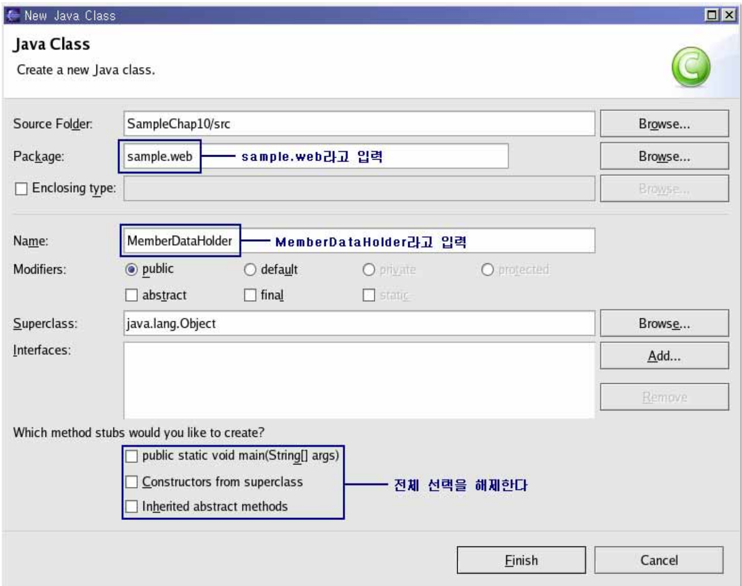


그림 10-21. Java클라스작성조수대화칸

MemberDataHolder.java가 작성되고 편집기가 현시되는데 여기에 목록 10-4와 같은 코드를 입력한다.(굵은 문자로 되여있는 부분은 추가 또는 수정한 코드이다.) 검색자료보존클라스는 종업원검색Servlet, 종업원검색클라스, 종업원검색결과 및 종업원검색오유화면들에서 취급하는 객체이고 열람기상에 표시하기 위한 종업원정보(ID, 이름, 주소) 및 검색결과통보창을 가진다. Getter 및 Setter메쏘드(get<실체변수이름>, set<실체변수이름>)는 차림표띠로부터 [Source]→[Getter 및 Setter의 생성]을 리용하여 쉽게 추가할수 있다.

목록 10-4. 검색자료보존클라스(MemberDataHolder.java)

```
/*
 * Created on Mar 25, 2005
 *
 * To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
package simple.web;
/**
 * @author Administrator
 *
 * To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
public class MemberDataHolder {
   private String id;
   private String name;
   private String address;
   private String message;
   public String getId() {
          return id;
   }
   public void setId(String id) {
          this.id = id;
   }
   public String getName() {
          return name;
   }
   public void setName(String name) {
          this.name = name;
   }
```

```
    public String getAddress() {
            return address;
    }
    public void setAddress(String address) {
            this.address = address;
    }
    public String getMessage() {
            return name;
    }
    public void setMessage(String message) {
            this.message = message;
    }
}
```

10.3.6. 종업원검색클라스의 작성

종업원검색클라스(MemberSearch.java)를 작성한다. 검색자료보존클라스와 같이 [Navigator]보기상에서 [SampleChap10]→[src]를 오른쪽찰칵하고 [New]→[Class]를 선택하면 Java클라스작성대화칸이 현시된다. 여기에서 [Package]에《sample.web》를 입력하고 [Name]에는 《MemberSearch》를 입력한다. [Modifiers]에서 [public]를 선택하고 [Superclass]에 《java.lang.Object》를 설정한 다음 [Which method stubs would you like to create ?]에서 모든 검사칸의 선택를 해제하고 [Finish]를 찰칵한다(그림 10-22).

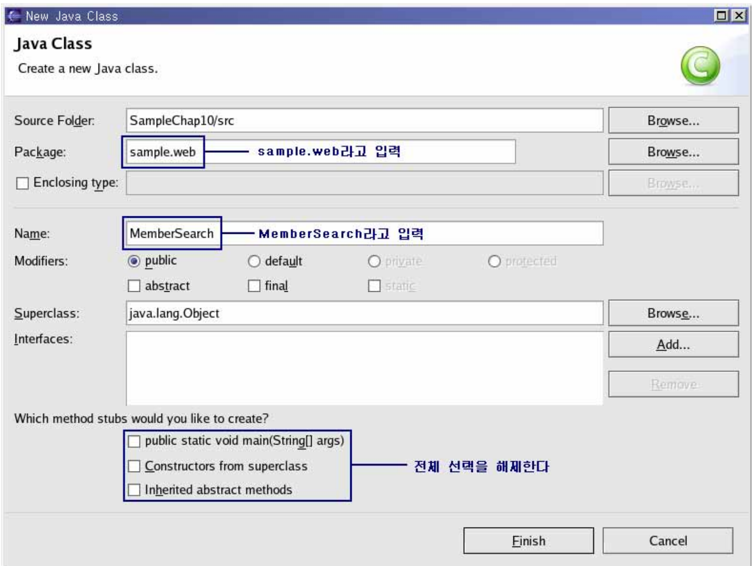


그림 10-22. Java클라스작성대화칸

MemberSearch.java가 작성되여 편집기에 현시되는데 여기에 목록 10-5의 코드를 입력한다.(굵은 문자로 되여있는 부분은 추가 또는 수정한 코드이다.) MemberSearch.java에서는 넘어온 종업원번호(memberId)를 가지고 우선 종업원목록(member.xml)에서 검색을 진행한다. 대응하는 종업원정보가 있는 경우에는 그 정보(종업원이름, 주소)및 검색이 성공하였다는것을 의미하는 통보문을 검색자료보존클라스(mdh)에 설정하고 귀환값으로서 1을 되돌린다. 또한 입력된 종업원번호에 대응하는 종업원정보가 없을 때에는 그 내용을 표현하는 통보문을 검색자료보존클라스(mdh)에 설정하고 -1을 귀환한다. 즉 MemberSearch.java에서는 XML형식의 종업원목록(member.xml)으로부터 정보를 검색하기 위하여 Java에서 XML문서를 처리하기 위한 API인 DOM(Document Object Model)을 리용하고있다.

목록 10-5. 종업원검색클라스(MemberSearch.java)

```
/*
 * Created on Mar 25, 2005
 *
 * To change the template for this generated file go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
package simple.web;

import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

/**
 * @author Administrator
 *
 * To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
public class MemberSearch {

 private static final String filePath =
   "c:/eclipse/workspace/MyTest/SimpleWeb/member.xml";
    private static final String msgInputError =
                "종업원ID가 입력되지 않았습니다.";
    private static final String msgSearchSuccess =
                "종업원정보 검색이 정확히 끝났습니다.";
    private static final String msgSearchError = "종업원정보가 틀립니다.";
    private static final String msgFailure =
                "종업원정보검색에서 장애가 발생했습니다.";
    private static NodeList memberList;
```

```
        static int search(MemberDataHolder mdh) {
          try {
             if(memberList == null) {
                DocumentBuilderFactory dbFactory =
                  DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = dbFactory.newDocumentBuilder();
                Document doc = builder.parse(new File(filePath));
                Element root = doc.getDocumentElement();
                memberList = root.getElementsByTagName("member");
             }

             String id = mdh.getId();

             if(id == null || id.equals("")) {
                mdh.setMessage(msgInputError);
                return -1;
             }

             for(int i=0; i<memberList.getLength(); i++) {
                 Element element = (Element) memberList.item(i);
                 if(id.equals(element.getAttribute("id"))) {
                     NodeList nameList = element.getElementsByTagName("name");
                     Element nameElement = (Element) nameList.item(0);
                     mdh.setName(nameElement.getFirstChild().getNodeValue());

                     NodeList addressList =
                          element.getElementsByTagName("address");
                     Element addressElement = (Element) addressList.item(0);
                     mdh.setAddress(addressElement.getFirstChild().
                          getNodeValue());

                     mdh.setMessage(msgSearchSuccess);
                     return 1;
                 }
             }
             mdh.setMessage(msgFailure);
             return -1;
          }
          catch(Exception e) {
             mdh.setMessage(msgFailure);
             return -1;
          }
        }
    }
```

다음으로 종업원목록(member.xml)을 작성한다. [Navigator]보기에서 [SampleChap10]→[SimpleWeb]를 선택한 상태에서 [New]→[File]을 선택하면 파일작성대화칸이 열린다. [File name]에 《member.xml》을 입력하고 [Finish]를 찰칵한다(그림 10-23).

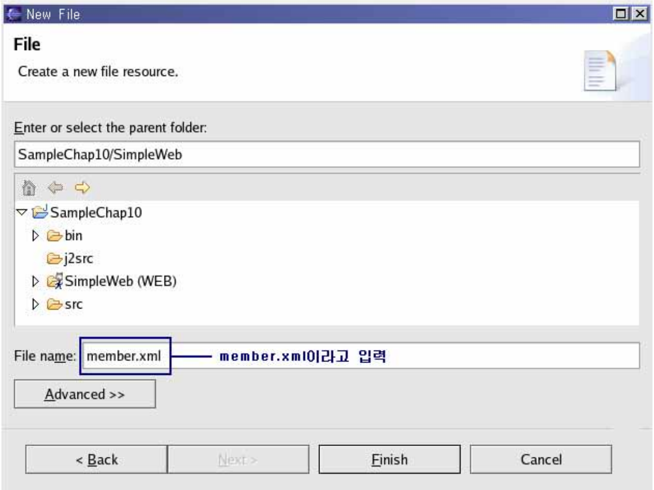


그림 10-23. 파일작성대화칸

member.xml(파일의 내용은 공백)이 작성되여 편집기에 현시되는데 여기에 목록 10-6의 코드를 입력시킨다.

목록 10-6. 종업원목록(member.xml)

```
<?xml version="1.0" encoding="UTF-8" ?>

<memberList>
   <title>Member List</title>
   <member id="1">
          <name>김철혁</name>
          <address>평양시 형제산구역 상당동 37반 1층 3호 </address>
   </member>
   <member id="2">
          <name>리철영</name>
          <address>평양시 락랑구역 승리1동 28반 3층 3호</address>
   </member>
   <member id="3">
          <name>박옥희</name>
          <address>평양시 모란봉구역 전우동 40반 5층 1호</address>
   </member>
</memberList>
```

10.3.7. 종업원검색결과화면(JSP)의 작성

종업원검색결과화면(MemberSearchOK.jsp)을 작성한다. [Navigator]보기에서 [SampleChap10]→[SimpleWeb]를 오른쪽찰칵하고 [New]→[Lomboz JSP Wizard]를 선택하면 JSP파일작성대화칸이 열린다. [File name]에 《MemberSearchOK》라고 입력하고 [Next]를 찰칵한다(그림 10-24).

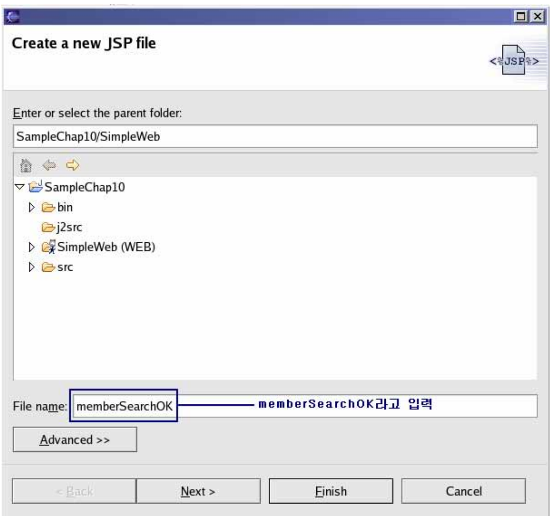


그림 10-24. JSP파일작성대화칸

[Set JSP details]대화칸에서 [Add]를 찰칵한다. [Select Beans]에서 [ID]에 《 mdh 》를 입력하고 [Scope]에서 《 request 》를 선택하며 [Select Class]에 《sample.web.MemberDataHoder》를 입력하고 [OK]를 찰칵한다(그림 10-25).

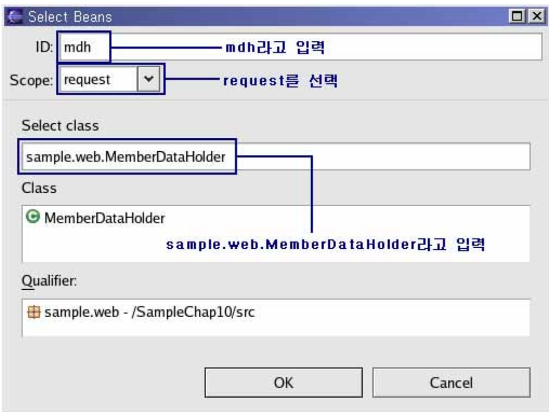


그림 10-25. 클라스선택대화칸

이러한 설정들을 진행하여 JSP안에서 검색자료보존클라스(MemberDataHoder)의 실체인 mdh를 취급할수 있게 된다. 다음에 [Set JSP details]대화칸의 [Content type]에서 《text/html;charset=UTF-8》, [Page encoding]에서 《UTF-8》을 선택한 다음 [Finish]를 찰칵한다(그림 10-26).

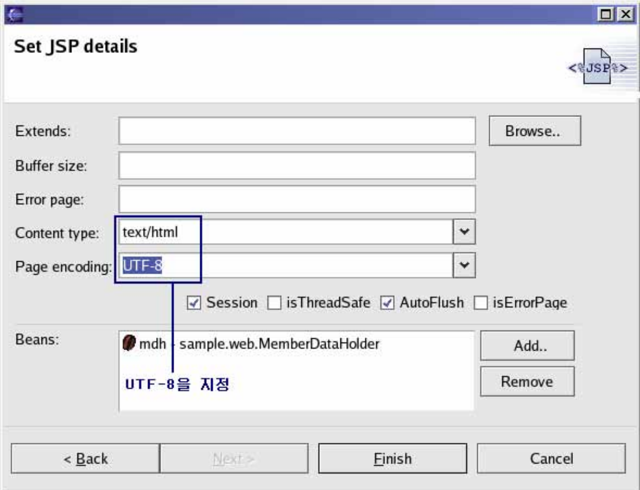


그림 10-26. JSP상세설정대화칸

이렇게 하면 memberSearchOK.jsp가 작성되고 편집기상에 작성된 코드가 현시된다. 목록 10-7과 같이 코드를 입력한다.(굵은 문자로 되여있는 부분은 추가 또는 수정한 코드이다.) MemberSearchOK.jsp에서는 mdh(종업원자료보존클라스의 실체)로부터 Getter메쏘드(get<실체변수이름>, 례 : getName)를 리용하여 정보를 얻고 화면에 현시하는 처리를 진행한다.

목록 10-7. 종업원검색결과화면(memberSearchOK.jsp)

```
<%@ page language="java" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>종업원검색결과</title>
</head>
<body bgcolor="#FFFFFF">
<jsp:useBean id="mdh" scope="request"
          class="simple.web.MemberDataHolder" />

<div align=" center" >
<h3>종업원검색결과</h3>
<%= mdh.getMessage() %>
```

```
<table width="500" cellpadding=10 align="center">
  <tr>
    <td width="20%" bgcolor="lightblue">종업원번호</td>
    <td width="80%" bgcolor="palegoldenrod"><%=mdh.getId() %></td>
  </tr>
  <tr>
    <td width="20%" bgcolor="lightblue">종업원이름</td>
    <td width="80%" bgcolor="palegoldenrod"><%=mdh.getName() %></td>
  </tr>
  <tr>
    <td width="20%" bgcolor="lightblue">주소</td>
    <td width="80%" bgcolor="palegoldenrod"><%=
          mdh.getAddress() %></td>
  </tr>
</table>
</div>
</body>
</html>
```

10.3.8. 종업원검색오유화면(JSP)의 작성

종업원검색오유화면(MemberSearchNG.jsp)을 작성한다. 종업원검색결과화면과 꼭같이 [Navigator]보기에서 [SampleChap10]→[SimpleWeb]를 오른쪽찰칵하고 [New]→[Lomboz JSP Wizard]를 선택하면 JSP파일작성대화칸이 나타난다. [File name]에 《memberSearchNG》를 입력하고 [Next]를 찰칵한다. [Set JSP details]대화칸에서 [Add]를 찰칵하고 [Select Beans]대화칸에서 [ID]에 《mdh》를 입력한다. 그리고 [Scope]에서 [request]를 선택하고 [Select class]에 《sample.web.MemberDataHolder》를 입력한 다음 [OK]를 찰칵한다. 이 설정들을 진행하여 JSP안에 검색자료보존클라스(MemberDataHolder)의 실체인 mdh를 취급할수 있게 된다.

다음으로 [Set JSP details(JSP상세설정)]대화칸의 [Content type(내용물형태)]에서 《text/html;charset=UTF-8》, [Page encoding(페지부호화)]에서 《UTF-8》을 선택한 다음 [Finish]를 찰칵한다.

그렇게 하면 memberSearchNG.jsp가 작성되고 편집기상에 작성된 코드가 현시된다. 목록 10-8과 같이 코드를 입력한다(굵은 문자로 되여있는 부분은 추가 또는 수정한 코드이다.). MemberSearchNG.jsp에서는 mdh로부터 오유통보문정보를 얻고 화면에 현시하는 처리를 진행한다.

이상으로 종업원검색응용프로그람의 개발은 완료된다.

목록 10-8. 종업원검색오유화면(memberSearchNG.jsp)

```
<%@ page language="java" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>종업원검색오유</title>
</head>
<body bgcolor="#FFFFFF">
<jsp:useBean id="mdh" scope="request"
    class="simple.web.MemberDataHolder" />

<div align="center">
<h3>종업원조사오유</h3>
<%= mdh.getMessage() %>
</div>

</body>
</html>
```

제4절. Web응용프로그람의 실행

작성한 종업원검색응용프로그람을 실행한다. 그러기 위해서는 Tomcat를 기동시켜 응용프로그람을 적재하여야 한다. 여기에서는 Lomboz의 기능을 사용하여 종업원검색응용프로그람의 전개 및 Tomcat의 기동을 진행하는 방법을 설명한다. 또한 응용프로그람의 실행 후 Tomcat의 정지를 진행하는 방법에 대하여 설명한다.

10.4.1. Web응용프로그람의 적재

차림표띠로부터 [Window]→[Show view]→[Lomboz J2EE View]를 선택한다. [Lomboz J2EE View]가 보이지 않는 경우에는 [Other]를 선택하고 [Lomboz J2EE]→[Lomboz J2EE View]를 선택한다. Lomboz J2EE View가 열리면 [SampleChap10]→[SimpleWeb]상에서 오른쪽찰칵하고 [Deploy(배치)]를 선택한다(그림 10-27). 그러면 그림 10-28과 같은 통보창이 [Console]보기에 현시되고 War(Web Application aRchive)파일의 작성 및 Tomcat에로의 적재가 시작된다.

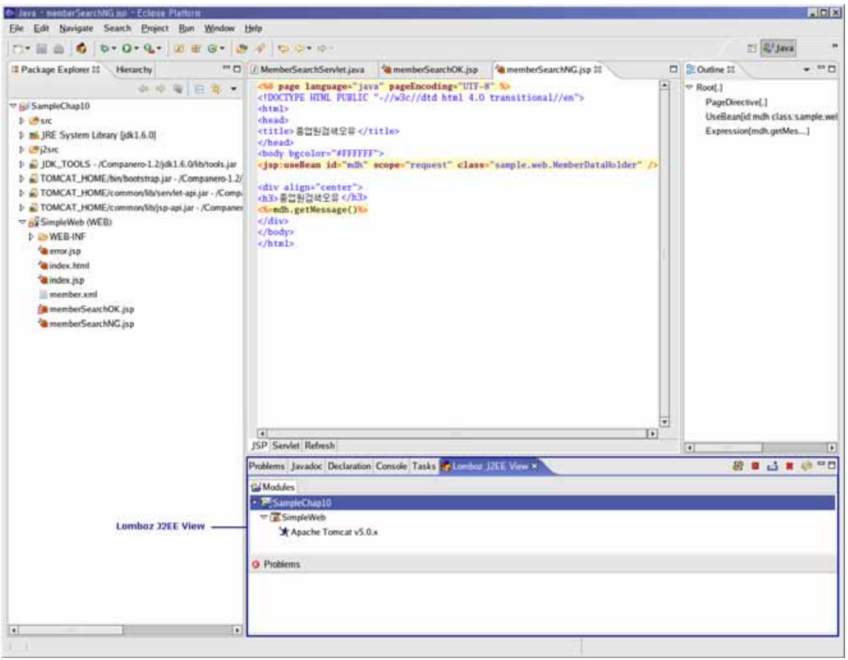


그림 10-27. 종업원검색응용프로그람의 적재

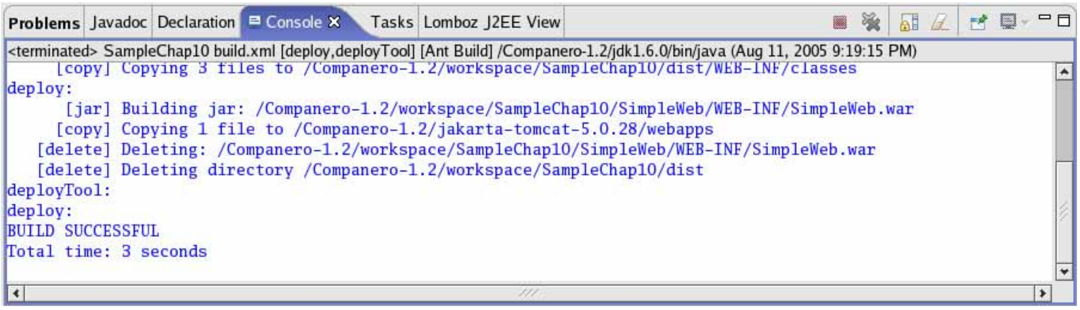


그림 10-28. 적재시 조종탁에 현시된 통보문

열람기를 기동하고 /kut/Tomcat/webapps등록부에 SimpleWeb.war가 배치되였는가를 확인한다. 즉 Lomboz에서는 War파일의 작성 및 Tomcat에로의 적재에 Ant가 리용되고있다. Ant건립파일(build.xml) 및 속성파일(build.properties)은 Lomboz에 의해 자동적으로 만들어지고 SampleChap10\SimpleWeb\WEB-INF아래에 배치되는데 이것을 확인해본다(목록 10-9, 10-10). 물론 [제8장 Ant지원]에서 설명한것처럼 [Navigator]보기에서 build.xml을 오른쪽찰칵하고 [Run]→[2 Ant build]를 선택한 후 [deploy]과제를 실행하는 등의 방법에 의해 War파일의 작성 및 Tomcat의 적재를 진행할수 있다.

목록 10-9. Lomboz에 의해 자동적으로 만들어진 Ant건립파일 (build.xml)

```
<!-- Copyright (c) 2002 by ObjectLearn. All Rights Reserved. -->
<project name="webmodulebuilder"  default="deploy" basedir=".">

  <!-- set global properties for this build -->
  <property file="build.properties"/>
  <property name="dist" value="../../dist" />
  <property name="web" value="../" />

  <target name="init">
    <!-- Create the dist directory structure used by compile
          and copy the deployment descriptors into it-->
    <mkdir dir="${dist}"/>
    <mkdir dir="${dist}/WEB-INF"/>
    <mkdir dir="${dist}/WEB-INF/classes"/>
    <mkdir dir="${dist}/WEB-INF/lib"/>
    <copy todir="${dist}">
      <fileset dir="${web}">
        <include name="**/*.*"/>
        <exclude name="**/jsp_servlet/*.class"/>
        <exclude name="**/build.xml"/>
        <exclude name="**/deploy.xml"/>
        <exclude name="**/build.properties"/>
        <exclude name="**/servers.xml"/>
        <exclude name="**/targets.xml"/>
        <exclude name="**/*.war"/>
      </fileset>
    </copy>
    <copy todir="${dist}/WEB-INF/classes">
      <fileset dir="${project.dir}/${bin.dir}">
        <include name="**/*.*"/>
        <exclude name="**/jsp_servlet/*.class"/>
      </fileset>
    </copy>

  </target>

  <target name="deploy" depends="undeploy,init" >
    <!  Create the distribution directory -->
    <delete file="${war}.war" failonerror="false" />
  <jar jarfile="${war}.war" basedir="${dist}"/>
```

```
        <copy file="${war}.war" todir="${deploy.dir}"/>
      <delete file="${war}.war" failonerror="false" />
      <delete dir="${dist}" failonerror="false" />
    </target>

    <target name="deployTool">
      <ant antfile="./deploy.xml" dir="." target="deploy" inheritall="true">
      </ant>
    </target>

    <target name="undeploy">
      <!-- Sometimes you can undeploy with deleting the module
        file but it is best dealt on an appserver basis
        at undeployTool target -->
    </target>
    <target name="undeployTool">
       <ant antfile="./undeploy.xml" dir="."
      target="undeploy" inheritall="true">
       </ant>
    </target>

  </project>
```

목록 10-10. Lomboz에 의해 자동적으로 만들어진 속성파일 (build.properties)

```
#LOMBOZ BUILD FILE PROPERTIES- UPDATED:Sat Mar 26 09:56:49 PST 2005
#Sat Mar 26 09:56:49 PST 2005
war=SimpleWeb
eclipse.home=/c\:/eclipse/
module=SimpleWeb
adminToolPath=
project.dir=../../
ejbsrc.dir=ejbsrc
deploy.dir=C\:/jakarta-tomcat-5.0.28/webapps
project.path=C\:\\eclipse\\workspace\\MyTest\\bin;C\:\\
      jakarta-tomcat-5.0.28\\bin\\bootstrap.jar;C\:\\
      jakarta-tomcat-5.0.28\\common\\lib\\servlet-api.jar;C\:\\
      jakarta-tomcat-5.0.28\\common\\lib\\jsp-api.jar
bin.dir=bin
```

10.4.2. Tomcat의 기동

다음에 Tomcat를 기동한다. [Lomboz J2EE View]를 열고 [SampleChap10]→[SimpleWeb]→[Apache Tomcat v5.0.x]상에서 오른쪽찰칵하고 [Run Server]를 선택한다(그림 10-29). 그림 10-30과 같은 통보문이 [Console]보기에 나타나고 Tomcat가 기동한다.

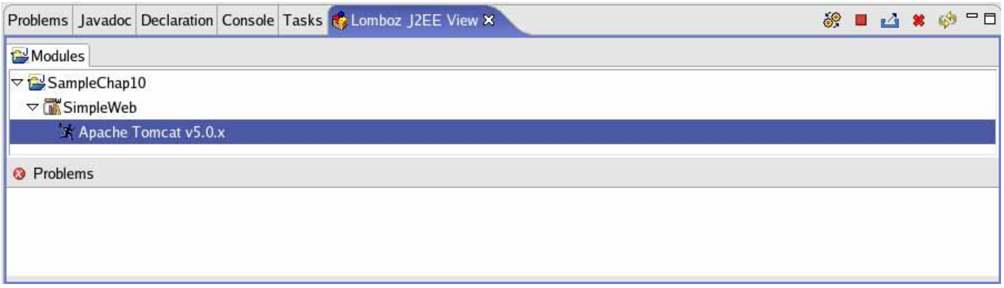


그림 10-29. Tomcat의 기동



그림 10-30. Tomcat기동할 때 조종탁통보문

열람기를 기동시키고 주소칸에《http://localhost:8080/SimpleWeb》를 입력한다. 그림 10-31과 같은 화면이 나타나면 성공이다.

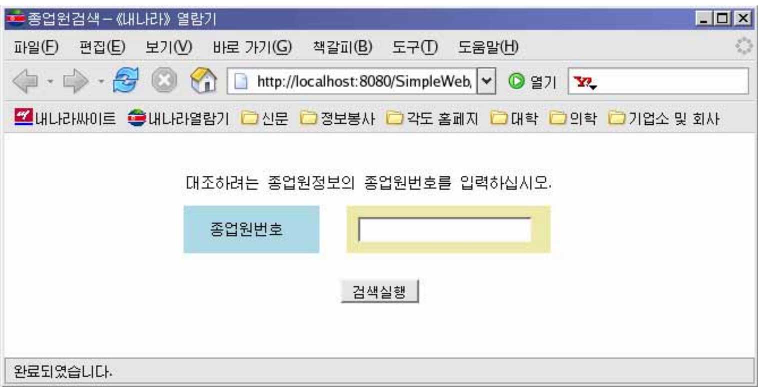


그림 10-31. 종업원검색화면

즉 Lomboz에서는 Tomcat의 기동에 Eclipse의 기동구성이 리용되고 있다. 차림표띠로부터 [Run]→[Run…]을 선택하면 [Create, manage and run configurations] 대화칸이 나타나는데 여기서 [Java Application]→[SampleChap10-Apache Tomcatv50x](Lomboz에 의해 자동적으로 설정된다.)를 선택하여 기동구성의 설정을 확인해본다(그림 10-32). 물론 [Create, manage and run configurations]대화칸에서 [Run]를 찰칵하여 Tomcat를 기동시킬수 있다.

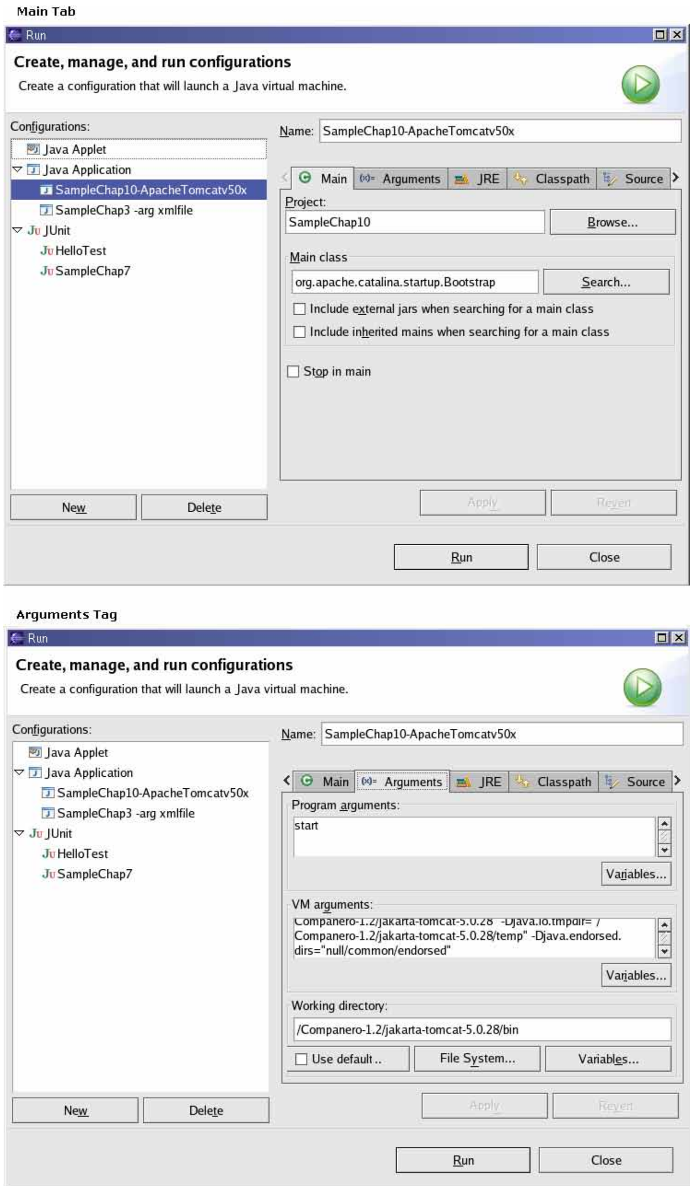


그림 10-32. Lomboz에 의해 자동설정된 Tomcat용 기동구성

10.4.3. Web응용프로그람의 실행

종업원검색화면의 종업원번호에 《1》을 입력하고 [검색실행]단추를 찰칵한다(그림 10-33).

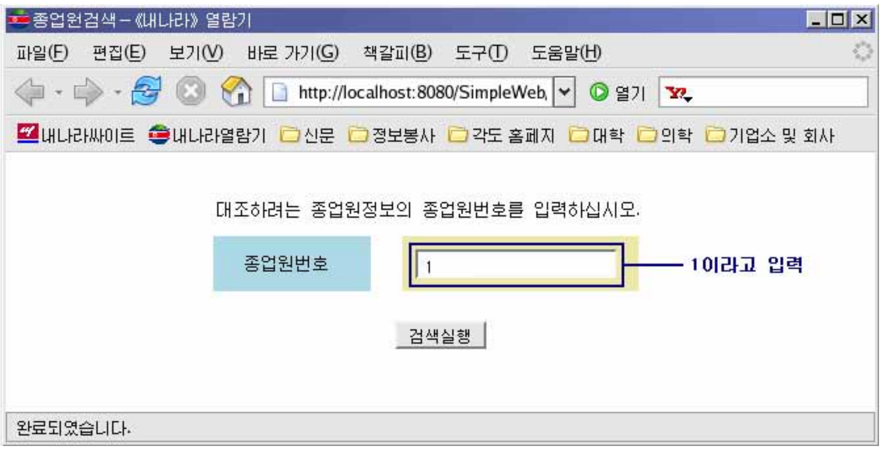


그림 10-33. 종업원검색화면에 자료입력

검색결과가 종업원검색결과화면에 나타난다(그림 10-34).

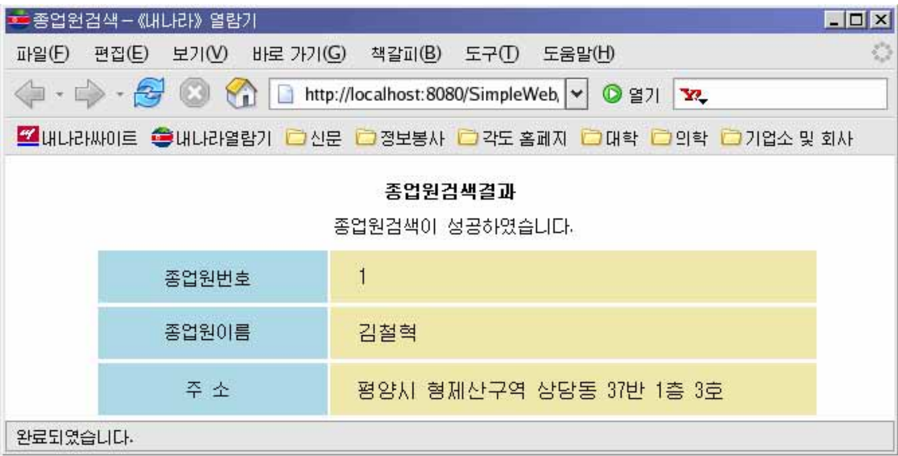


그림 10-34. 종업원검색결과화면

열람기상의 [Back]단추를 찰칵하고 다시한번 종업원검색화면을 펼친다. 이번에는 《5》를 입력하고 [검색실행]단추를 찰칵한다. 그러면 그림 10-35과 같이 종업원검색오유화면이 현시된다.

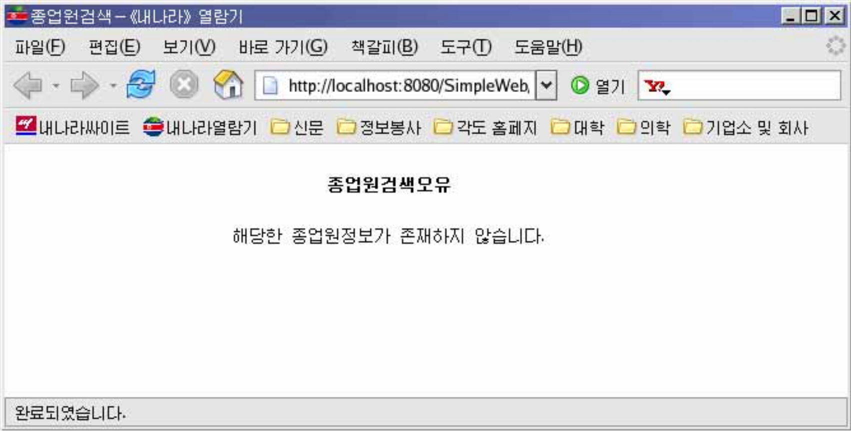


그림 10-35. 종업원검색오유화면

이상으로 종업원검색응용프로그람의 실행이 확인되였다.

10.4.4. Tomcat의 정지

Tomcat를 정지시키기 위하여 [Lomboz J2EE View]로부터 [SampleChap10]→[SimpleWeb]→[Apache Tomcat v5.0.x]를 선택한 상태에서 오른쪽찰칵하여 [Stop Server]를 선택한다. 그림 10-36과 같은 통보문이 [Console]보기에 나타나고 Tomcat가 정지된다.

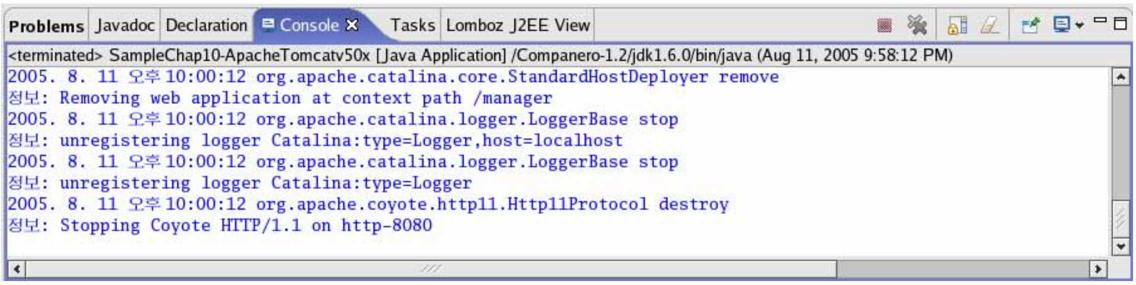


그림 10-36. Tomcat정지시 조종탁의 통보문

제5절. Web응용프로그람의 오유수정

Eclipse 및 Lomboz의 기능에 의해 Web응용프로그람의 통합적인 오유수정환경이 제공된다. 여기서는 Servlet 및 JSP의 오유수정방법에 대하여 설명한다. Servlet의 오유수정 례로서 종업원검색Servlet(MemberSearchServlet.java), JSP의 오유수정으로서 종업원검색결과화면(memberSearchOK.jsp)을 취급한다.

10.5.1. 국부적인 오유수정방법과 원격오유수정방법

Eclipse 및 Lomboz를 리용하면 Web응용프로그람의 국부적인 오유수정 및 원격오유수정 둘다 가능하다.

여기서 말하는 **국부적인 오유수정**이란 Eclipse상에서 Tomcat를 기동(Eclipse가 Tomcat를 직접조종)하고 오유수정를 하는것을 말한다. Eclipse가 가동하는 국부콤퓨터상에 Tomcat가 존재하여야 한다.

한편 **원격오유수정**이란 국부콤퓨터상에서 가동하는 Eclipse로부터 J2SE SDK의 JPDA(Java Platform Debugger Architecture)를 통하여 Eclipse와는 독립적으로(보통은 원격콤퓨터상에서) 기동되고있는 Tomcat에 대한 접속, 감시를 진행하는것으로 오유수정를 하는것을 말한다.(이 책에서는 편리상 국부콤퓨터상에서 Tomcat를 기동하고 다른 콤퓨터에서 국부콤퓨터를 지정하여 접속을 진행한다.)

10.5.2. 종업원검색Servlet의 국부적인 오유수정

종업원Servlet를 실례들어 Servlet의 국부적인 오유수정에 대하여 설명한다.

1. 멈춤점의 설정

종업원검색servlet의 MemberSearchServlet에 멈춤점을 설정한다.

[Navigator]보기에서 MemberSearchServlet.java를 선택하여 편집기에 그 파일을 현시한다. 멈춤점을 설정하려는 행(이 례에서는 31행)의 왼쪽끝에 마우스유표를 이동하고 두번 찰칵하든지 아니면 오른쪽찰칵하여 [Toggle Line Breakpoint]에서 설정한다.(그림 10-37)

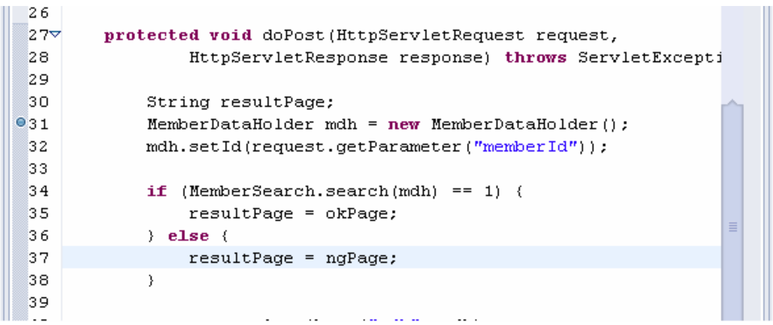


그림 10-37. 멈춤점의 설정

2. 오유수정의 실행

오유수정을 진행하기 위하여 Tomcat를 오유수정방식으로 기동시킨다. [Lomboz J2EE View]보기에서 [SampleChap10]→[SimpleWeb]→[Apache Tomcat v5.0.x]를 오른쪽찰칵하고 [Debug Server]를 선택한다([Run Server]에서 기동한 경우에는 한번 [Stop Server]로 정지되기때문에 [Debug Server]로 재기동한다. 이미 [Debug Server]로 기동중이라면 재기동할 필요가 없다.). 이때 그림 10-38과 같은 통보창이 [Console(조종탁)]보기상에 현시되고 Tomcat가 기동한다.

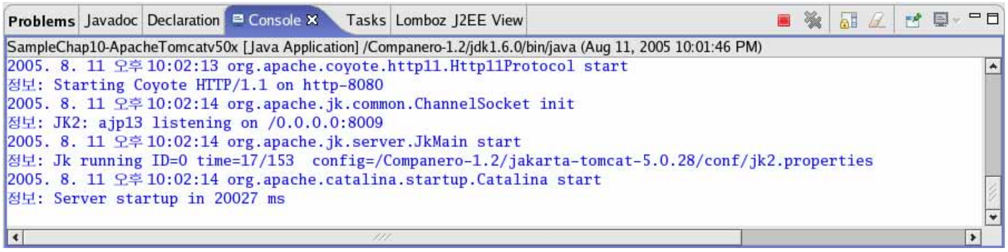


그림 10-38. Tomcat가 기동할 때 조종탁통보문

열람기를 기동하여 주소칸에《http://localhost:8080/SimpleWeb》를 입력하면 종업원검색화면이 나타난다. 종업원번호에 《1》을 입력하고 [검색실행]단추를 누른다. 화면이 [Debug]투시로 절환되고 멈춤점을 설정한 행에서 실행이 정지된다.

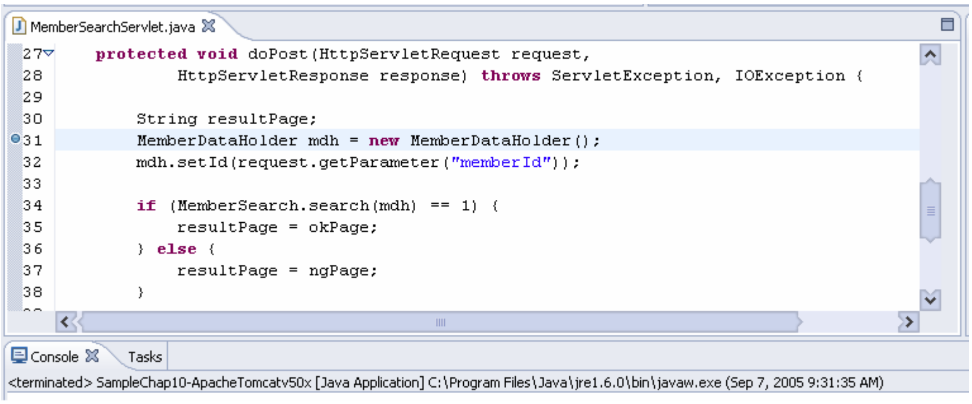


그림 10-39. [Debug]투시

[Debug]투시로 절환한 경우는 [Remember my decision]를 선택하고 [Ok]를 찰칵한다.(이후부터 이 대화칸은 표시되지 않는다.)

화면왼쪽상의 [Debug]보기의 도구띠에 배치된 단추 또는 [Run]차림표를 리용하

여 [Step Into], [Step Over],[Step Return] 등을 시험해본다. (오유수정의 자세한 부분은 [제5장 오유수정]를 참조) [Debug]보기에서 [Resume]단추를 누르는 조작에 의해 마지막까지 처리가 실행되면 열람기에 실행결과가 현시된다.

이상에서 종업원Servlet의 국부적인 오유수정에 대한 설명은 끝난다. 실행결과를 확인하려면 [Lomboz J2EE View]로부터 [SampleChap10]→[SimpleWeb]→[Apache Tomcat v5.0.x]를 선택한 상태에서 오른쪽찰칵하여 [Stop Server]를 선택하여 Tomcat를 정지시킨다.

10.5.3. 종업원검색Servlet의 원격오유수정

앞에서와 마찬가지로 종업원검색Servlet를 실례들어 Servlet의 원격오유수정에 대하여 설명한다.

1. Tomcat의 기동서술의 편집과 기동

원격오유수정를 진행하는데는 Tomcat를 원격오유수정기능을 설정하고 기동시켜야 한다. Tomcat를 이미 기동시키고있었다면 정지시켜야 한다.

《\Companero-1.2\jakarta-tomcat-5.0.28\bin》안에 startup.sh라는 기동서술파일을 별명(startup_debug.sh)으로 복사하고 이 파일을 편집기에서 열어 아래와 같이 한 행을 추가한다.

```
Set CATALINA_OPTS=-Xdebug -Xnoagent -Djava.compiler=NONE
                  -Xrunjdwp:transport=dt_socket,server=y,
                   suspend=n
```

작성한 startup_debug.bat를 편집기상에서 두번찰칵하여 Tomcat를 기동시킨다. 그러면 그림 10-40과 같이 기동화면이 나타난다.

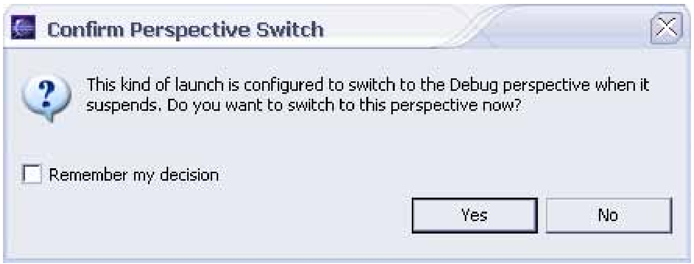


그림 10-40. 기동화면

기동화면에 추가된 통보의 머리부에 아래와 같은 행이 있는것을 볼수 있다.

Listening for transport dt_socket at address: XXXX

여기서 XXXX는 포구번호로서 4자리의 수자가 입력된다. 후에 리용하기때문에 변경시킨다.

Catalina. bat의 기능항목에 의한 설정

《\Companero-1.2\jakarta-tomcat-5.0.28\bin》안의 catalina.sh기능을 리용하여 아래와 같은 설정을 진행한다.

Startup_debug. s h 의 마지막에 [call "%EXECUTABLE%" start %CMD_LINE_ARGS%]라는 행과 그전에 아래와 같이 대문자로 쓴 내용을 추가한다. JPDA_ADDRESS에는 임의의 포구번호를 지정하여 쓸수 있다.

```
Set JPDA_TRANSPORT=dt_socket
Set JPDA_ADDRESS=8888
Call "%EXECUTABLE%" jpda start %CMD_LINE_ARGS%
```

2. 멈춤점의 설정

[10.5.2. 종업원검색Servlet의 국부적인 오유수정]의 [멈춤점의 설정]에서 실시한것과 같은 방법으로 종업원검색Servlet(MemberSearchServlet.java)에 멈춤점을 설정한다.

3. 원격오유수정용 기동구성의 설정

오유수정를 실행하기 전에 원격오유수정용의 기동구성에 대한 설정을 진행한다.

Eclipse상의 차림표띠로부터 [Run]→[Debug]에서 [Create, manage, and run configurations]대화칸을 열고 대화칸의 왼쪽에서 [Remote Java Application]을 선택하고 아래쪽에 있는 [New]를 찰칵한다.

4. 오유수정의 실행

열람기를 기동하여 주소칸에《http://localhost:8080/SimpleWeb》를 입력하면 종업원검색화면이 나타난다. 종업원번호에 《1》을 입력시킨 다음 [검색실행]단추를 찰칵한다. 그러면 Eclipse의 화면이 [Debug]투시로 절환되고 멈춤점을 설정한 행에서 중지한다(그림 10-41).

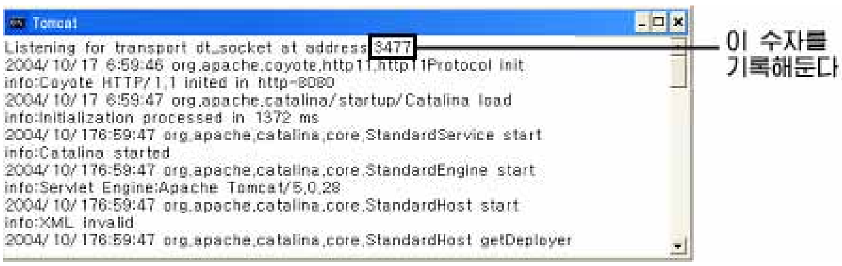


그림 10-41. debug방식으로 Tomcat기동

화면왼쪽상의 [Debug]보기의 도구띠에 배치된 단추를 리용하여 걸음실행을 해본다. [Replay]단추를 찰칵하여 마지막까지 실행하면 열람기에 실행결과가 나타난다.

이상으로 종업원검색Servlet의 원격오유수정에 대한 설명을 끌마친다.

열람기로부터 《\Companero-1.2\jakarta-tomcat-5.0.28\bin》등록부에 있는 shutdown.bat를 두번찰칵하여 Tomcat를 중지시킨다.

10.5.4. 종업원검색결과화면(JSP)의 국부적인 오유수정

여기서는 종업원검색결과화면을 실례들어 JSP의 국부적인 오유수정에 대하여 설명한다.

1. 상태설정파일의 설정과 클라스파일의 복사

JSP의 오유수정에서는 JSP파일에 직접 멈춤점을 설정할수 없다.

Tomcat의 JSP엔진은 Servlet원천파일을 리용한다. Eclipse안에서 멈춤점을 설정하자면 JSP Servlet원천파일을 프로젝트에 넣어야 한다.

우선 Tomcat에 대한 설정을 진행한다. Tomcat가 기동되여있다면 정지시킨다. JSP Servlet원천파일을 생성하고 특정한 등록부에 배치하자면 상태설정파일의 설정을 진행해야 한다. 열람기를 기동하고 《\Companero-1.2\jakarta-tomcat-5.0.28\conf\Catalina\localhost》에 《SimpleWeb.xml》이라는 이름을 가진 파일을 새로 작성한다. 그리고 작성한 SimpleWeb.xml파일을 편집기에 불러들여 아래와 같이 서술해놓는다. [docBase]는 문맥의 뿌리등록부를 표시한다. 결국 이 설정에 의해 문맥의 뿌리등록부가 《\Companero-1.2\jakarta-tomcat-5.0.28\webapp\SimpleWeb》로부터 《\kut\workspace\SampleChap10\SimpleWeb》로 변경된다. 또한 [workDir]는 JSP Servlet원천파일의 생성장소를 표시한다.

```
<Context path= “/SimpleWeb”
   docBase= “\KUT\workspace\SampleChap10\SimpleWeb”
   workDir= “\KUT\workspace\SampleChap10\j2src”  />
```

[docBase]에서 설정한 《\kut\workspace\SampleChap10\SimpleWeb》등록부아래에 있는 《WEB-INF\classes》에는 종업원검색응용프로그람의 실행에 필요한 클라스파일(MemberDataHolder.class, MemberSearchServlet.class, MemberSearch.class)들이 존재하지 않기때문에 《SampleChap10\bin》아래에 존재하는 클라스파일을 복사해놓아야 한다. [Navigator]보기에서 [SampleChap10]→[bin]→[sample]을 선택하고 오른쪽찰칵한 다음 [Copy], [Paste]를 리용하여 파일의 복사를 진행한다.

2. Tomcat의 기동과 JSP Servlet의 원천파일의 넣기

[Lomboz J2EE View]보기에서 [SampleChap10]→[SimpleWeb]→[Apache Tomcat v5.0.x]를 오른쪽찰칵하여 [Debug Server]를 선택한다. 그러면 Tomcat가 기동한다. 열람기를 기동시켜 《http://localhost:8080/SimpleWeb》로 종업원검색화면을 열고 [종업원번호]에 《1》을 입력한 다음 [검색실행]단추를 찰칵한다.

그다음 [Navigator]보기에서 [SampleChap10]을 오른쪽찰칵하고 [Refresh]을 선택한다. 그러면 [j2src]등록부아래에 [org\apache\jsp]등록부가 생기고 《memberSearchOK_jsp.java》파일이 나타난다. 즉 작성된 JSP Sevlet원천파일에 콤파일오유가 발생하지만 오유수정에는 지장이 없다.

아래에서 서술한 [JSP Servlet원천파일을 취급할 때에 발생하는 콤파일오유에 대하여]를 참고한다.

3. 멈춤점의 설정

memberSearchOK_jsp.java에 멈춤점을 설정한다.

먼저 [Navigator]보기상에서 memberSearchOK_jsp.java를 선택하고 편집기를 연다. 메쏘드안의 적당한 위치에 멈춤점을 설정한다.(이 목록에서는 31행) 편집기의 왼쪽끝에서 오른쪽찰칵하고 [Toggle Method Load Breakpoint]을 선택한다(그림 10-42).

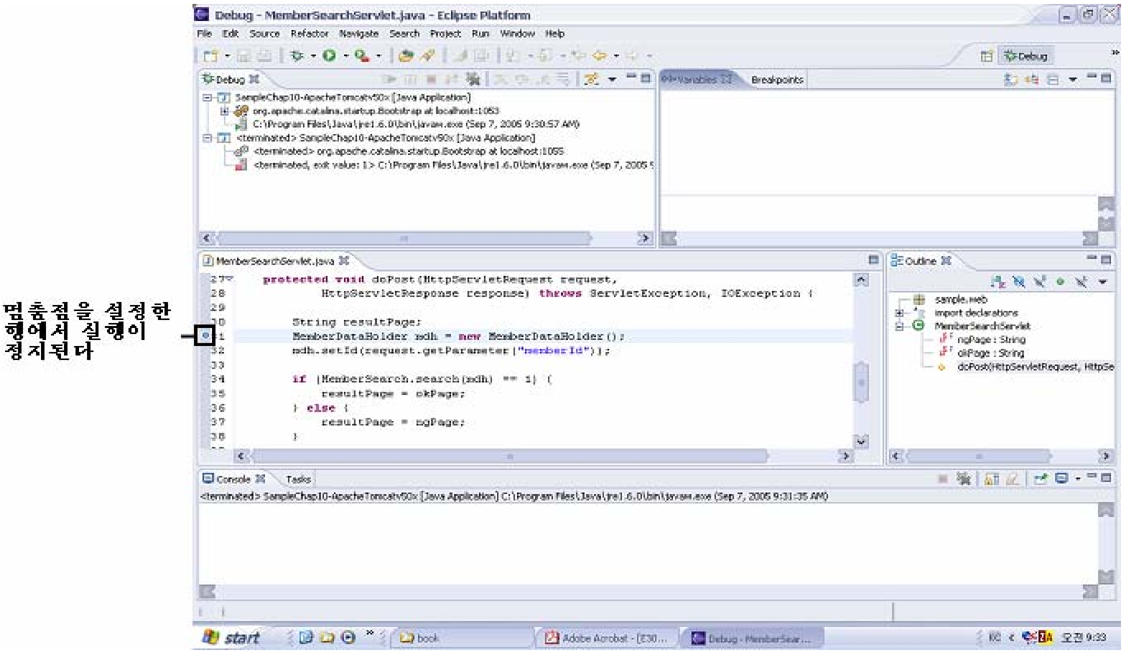


그림 10-42. 멈춤점의 설정

4. 오유수정의 실행

memberSearchOK_jsp.java의 오유수정을 진행한다. 열람기를 기동하여 《http://localhost:8080/SimpleWeb》으로 종업원검색화면을 열고 [종업원번호ID]에 《1》을 지정한 다음 [검색실행]단추를 찰칵한다. 화면이 [Debug]투시로 절환되고 멈춤점을 설정한 행에서 실행이 정지된다(그림 10-43).

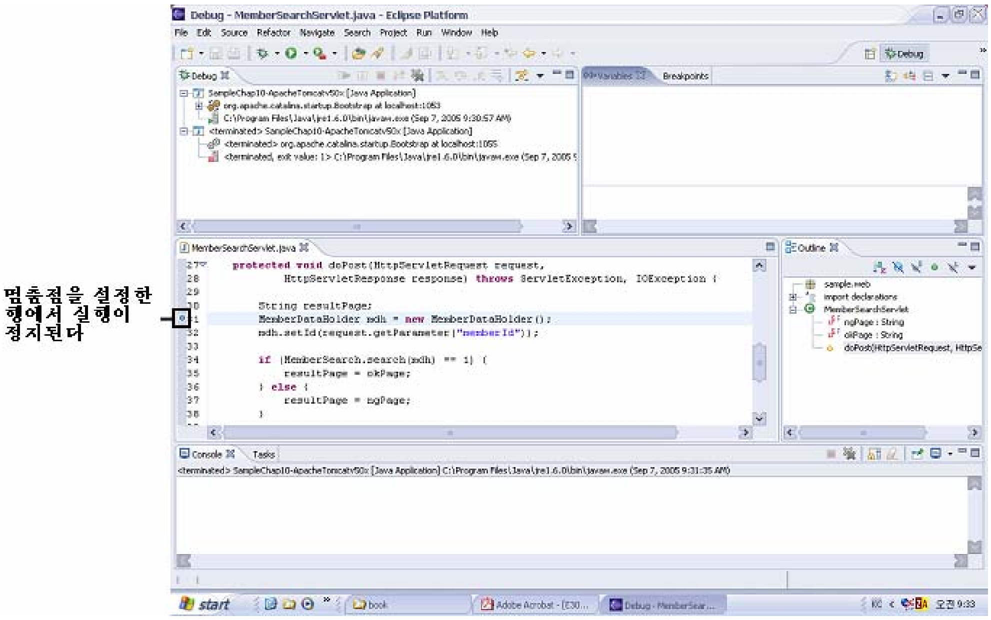


그림 10-43. [Debug]투시

멈춤점에서 정지가 잘 되지 않는 경우에는 Tomcat를 일단 끝내고 다시 한번 Debug Server로서 기동한다. 화면왼쪽상의 [Debug]보기의 도구띠에 배치된 단추를 리용하여 단계실행을 진행한다. [Replay]단추를 찰칵하여 마지막까지 처리가 실행되면 열람기에 실행결과가 현시된다.

이상으로 종업원검색결과화면(JSP)의 국부적인 오유수정에 대한 설명을 끝마친다. 실행결과를 확인한 다음 [Lomboz J2EE View]로부터 [SampleChap10]→[SimpleWeb]→[Apache Tomcat v5.0.x]를 선택하고 오른쪽찰칵하여 [Stop the launched Server]를 선택하고 Tomcat를 정지시킨다.

JSP Servlet원천파일을 취급할 때 발생하는 콤파일오유에 대하여

JSP Servlet원천파일(xxx_jsp.java)을 Eclipse에서 취급할 때 콤파일오유 및 표시할 때 문자이지러짐이 생긴다. 이것은 클라스경로에 서고(jasper-runtime.jar)가 맞지 않는것, 그리고 Tomcat가 JSP Servlet원천파일을 Unicode(UTF-8)로 출력하는데 대해 Eclipse작업대의 기정부호화는 MS932로 되고있기때문에 나타나는 현상이다. JSP Servlet원천파일을 수정하지 않아도 오유수정은 문제없이 진행되지만 콤파일오유를 없애버리자면 다음의 방법을 리용해야 한다.

△ 클라스경로의 jasper-runtime.jar의 추가

[Navigator]보기상에서 [SampleChap10]을 선택하고 오른쪽찰칵하여 [Properties]를 선택하면 프로젝트의 [Properties]대화칸이 열린다. [Java Working Path]를 선택하여 대화칸의 오른쪽의 [Library]태브를 선택하고 [Add External JAR]를 선택한다. 《/opt/Tomcat/common/lib/jasper-runtime.jar》를 선택하고 [Open]단추로 추가한다. 다음 [OK]단추를 찰칵하여 대화칸을 닫는다.

△ 부호화문제의 처리

해당한 JSP Servlet원천파일이 선택된 상태에서 차림표띠로부터 [Edit]->[Encoding]->[UTF-8]을 선택하고 파일의 부호화를 UTF-8로 변환한다. 다음 JSP Servlet원천파일을 수정하고 보존할수 있는 상태로 한다. 그리고 보관 및 건립을 진행한다. 그러면 JSP Servlet원천파일의 콤파일오유 및 문자이지러짐은 해소된다.

10.5.5. 종업원검색결과화면(JSP)의 원격오유수정

여기에서는 종업원검색결과화면을 실례들어 JSP의 원격오유수정에 대하여 설명한다.

1. 오유수정의 준비

[10.5.4. 종업원검색결과화면(JSP)의 국부적인 오유수정]에서 설명한 상태설정파일의 설정, 클라스파일의 복사 및 JSP Servlet원천파일의 취급이 진행된것을 확인한다.

2. Tomcat의 기동

[10.5.3. 종업원검색Servlet의 원격오유수정]의 [Tomcat의 기동서술의 편집과 기동]에서 작성한 startup_debug.bat를 리용하여 Tomcat(오유수정항목이 부가된)를 기동한다. 기동화면에서 표시되는 4자리의 포구번호를 변화시킨다(그림 10-44).

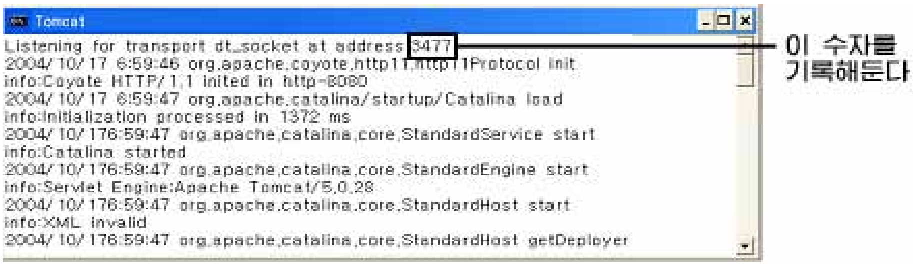


그림 10-44. 오유수정방식으로 Tomcat기동

3. 멈춤점의 설정

[10.5.4. 종업원검색결과화면(JSP)의 국부적인 오유수정]의 [멈춤점의 설정]에서와 같은 방법으로 종업원검색결과화면의 JSP Servlet원천파일 (memberSearchOK_jsp.java)에 멈춤점을 설정한다(그림 10-45).

4. 원격오유수정시 기동구성의 설정

Eclipse상의 차림표띠로부터 [Run]→[Debug]에서 [Create, manage, and run configurations]대화칸을 열고 대화칸왼쪽에서 [10.5.3 종업원검색Servlet의 원격오유수정]의 [원격오유수정용의 기동구성의 설정]에서 등록한 [Remote Java Application]→[SampleChap10]을 선택한다. [Connection Properties]의 포구로 선정변환한(또는 기동선택항목안에서 address=XXXX로 지정한) 4자리의 수자를 입력한다. 그림 10-46과 같이 설정된다.

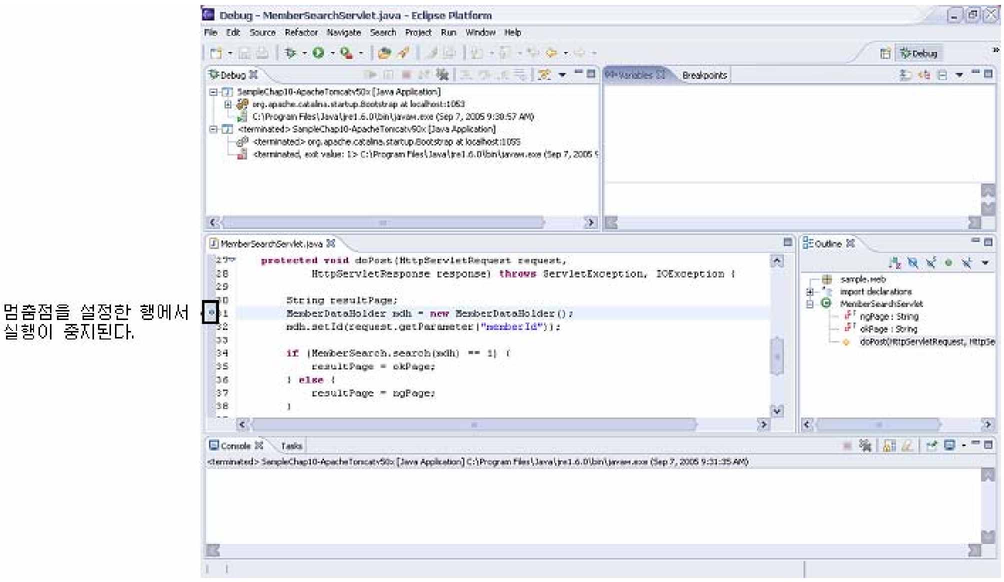


그림 10-45. [Debug]투시

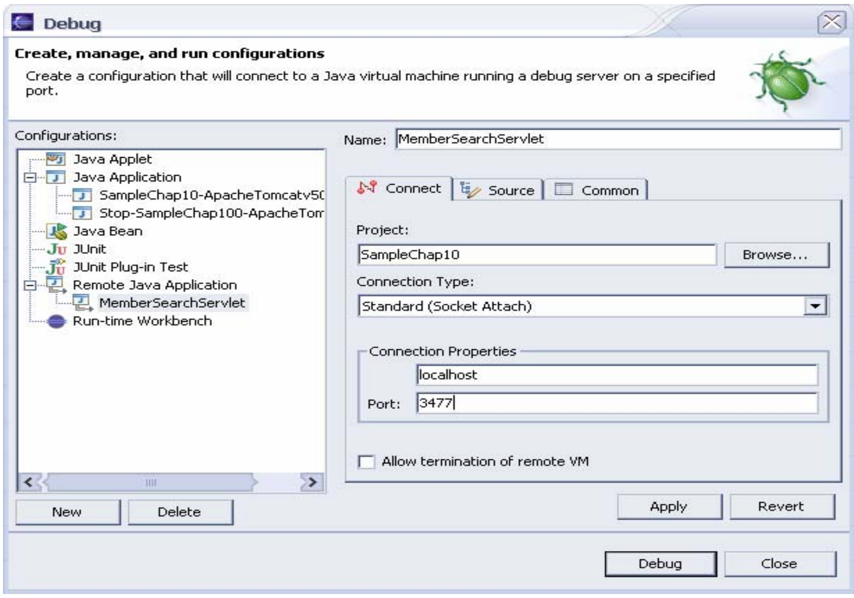


그림 10-46. 원격오유수정용 기동구성의 설정

[Apply]단추를 찰칵한다. 다음 [Debug]단추를 찰칵하고 Tomcat에 원격접속한다. 그러면 오유수정의 실행이 가능하게 된다.

5. 오유수정의 실행

열람기를 기동하여 《http://localhost:8080/SimpleWeb》로 종업원검색화면을 현시하고 종업원번호에 《1》을 입력한 다음 [검색실행]단추를 찰칵한다. 그러면 Eclipse의 화면이 [Debug]투시로 절환되고 멈춤점을 설정한 행에서 정지된다(그림 10-47).

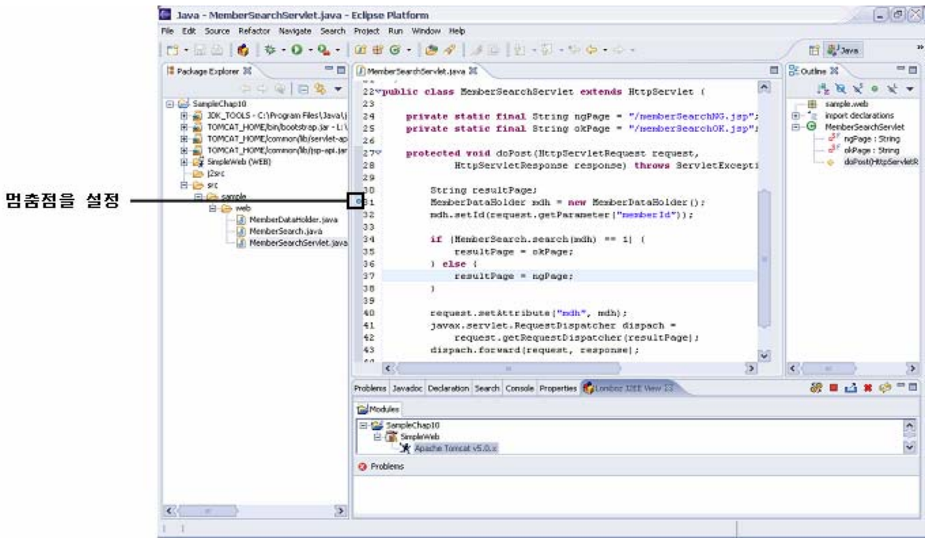


그림 10-47. 멈춤점의 설정

화면왼쪽우의 [Debug]보기의 도구띠에 배치된 단추를 리용하여 걸음실행을 진행한다. [Replay]단추를 찰칵하면서 마지막까지 처리가 실행되면 열람기에 실행결과가 표시된다.

이상으로 종업원검색결과화면(JSP)의 원격오유수정에 대한 설명을 끝마친다. 열람기로부터 《/opt/Tomcat/bin》에 있는《shutdown.sh》를 두번찰칵하여 Tomcat를 정지시킨다. JSP의 오유수정표준작업에서 설정한 《SimpleWeb.xml》파일을 지우고 복사해둔 [SampleChap10/SimpleWeb/WEB-INF/classes]아래의 클라스파일도 지워버린다.

제11장. JBoss-IDE에 의한 J2EE개발

제1절. JBoss와 JBoss-IDE

11.1.1. JBoss

JBoss는 현재 크게 주목되고있는 공개원천의 J2EE응용봉사기이다. 최신판본(ver3.2.5)에서는 Tomcat5.0.26, JSP 1.2, Servlet 2.3, EJB 2.0, JCA, JTA, JMX 등을 포함한 J2EE 1.3의 거의 모든 기능을 지원하고있으며 대규모체계에서 없어서는 안될 무리짓기기능(clustering)까지 부여하고있다. 허가료금을 물 필요가 없으며 상업용으로 리용하는 경우에도 무료로 쓸수 있다.

이 장에서는 JBoss와 Eclipse를 련동시켜 J2EE응용프로그람의 실행 및 오유수정를 진행하는 방법에 대하여 설명한다. Windows판본에 기초하여 설명하기때문에 Linux에서 쓸 때에는 해당한 설정을 진행하여야 한다.

11.1.2. JBoss-IDE

JBoss-IDE는 J2EE개발지원을 위한 Eclipse의 끼워넣기이다. 1.3.30판본에서는 다음과 같은 기능을 제공하고있다.

- XDoclet의 지원
- JBoss봉사기의 시동과 정지
- JBoss봉사기상에서의 오유수정 및 감시
- 보존자료(또는 적재한 등록부로서의)의 패키지화배치를 형성하는 간단한 방법
- J2EE요소를 JBoss봉사기에 배치하기 위한 기능
- J2EE요소의 개발조수

여기서는 JBoss-IDE를 사용한 EJB의 개발방법에 대하여 소개한다.

11.1.3. XDoclet

XDoclet는 공개원천인 코드생성엔진이다. XDoclet는 원천코드를 해석하고 JavaDoc에 서술되여있는 특별한 서술자(@web.servlet, @jboss.ejb-ref 등)에 기초하여 서술문서(web.xml, ejb-jar.xml 등)나 원천(EJB기본대면, EJB원격대면 등)을 자동생성한다. XDoclet는 Ant의 과제로서 실행된다. XDoclet를 사용하면 개발에서 생산성을 향상시킬수 있다.

제2절. 설치와 설정

JBoss와 JBoss-IDE는 《http://www.jboss.org》로부터 무료로 내리적재할수 있다.

11.2.1. JBoss의 설치

JBoss의 설치는 다음과 같은 순서로 진행한다.

1. 먼저 J2SDK를 설치한다.
2. jboss-3.2.5.zip를 리용하여 JBoss를 설치한다. Jboss-3.2.5.zip를 적당한 등록부(여기서는 C:/KUT)에 풀어놓는다.
3. 다음으로 환경변수를 설정한다.

설정할 환경변수는 표 11-1과 같다.

표 11-1. 설정할 환경변수

환경변수	경 로
JAVA_HOME	J2SDK의 설치등록부(여기서는 C:/KUT/j2sdk)
PATH	%JAVA_HOME%/bin
JBOSS_HOME	JBoss의 설치등록부(여기서는 C:/KUT/jboss-3.2.5)

4. 다음으로 JBoss의 기동을 확인한다. 기동은 《%JBOSS_HOME%/bin》등록부에 있는 run.bat를 실행하여 실행한다.
5. 기동하면 그림 11-1과 같은 지령입력재촉화면이 나타난다.

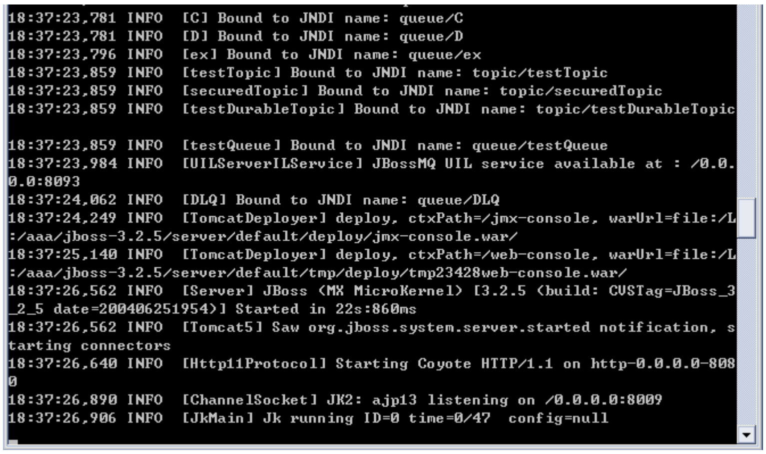


그림 11-1. JBoss기동화면

열람기를 기동시키고 《http://localhost:8080/web-console/》에 접근한다. 그림 11-2와 같은 화면이 나타나면 정확히 설치가 끝났다는것을 의미한다.

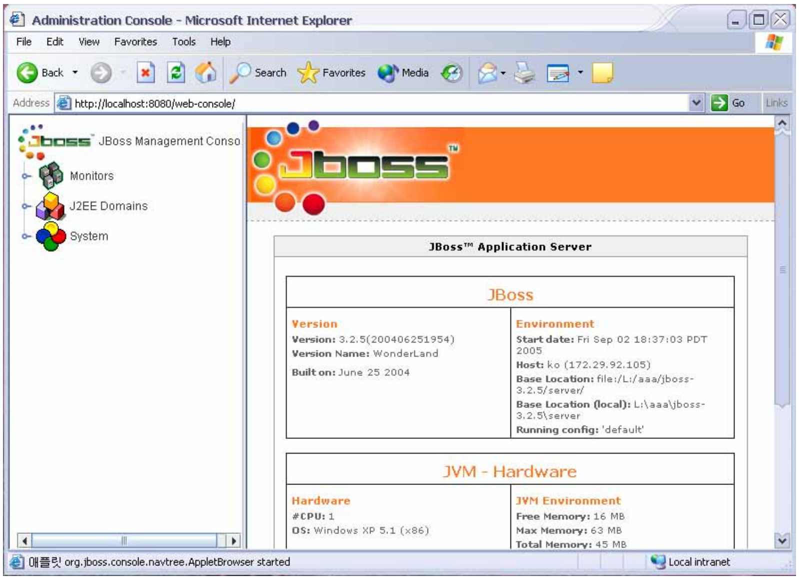


그림 11-2. JBoss Web조종탁

봉사기를 정지하려면 지령입력재촉화면에 다음과 같이 써넣으면 된다.

C:/KUT/jboss-3.2.5/bin/shutdown.bat -S

또한 그림 11-1의 지령입력재촉상에서 Ctrl + C를 입력해도 봉사기를 정지시킬수 있다.

11.2.2. JBoss-IDE의 설치

JBoss-IDE의 설치는 다음과 같은 순서로 진행한다.

1. Eclipse를 기동한다.
2. [Help]→[Software Updates]→[Find and Install]를 선택한다.
3. [Search for new features to install]을 선택하고 [Next]단추를 찰칵한다. [New Remote Site]를 클릭하여 현시되는 대화칸의 [Name]에 《JBoss-IDE》를 입력하고 [URL]에《http://jboss.sourceforge.net/jbosside/updates》를 입력한 다음 [OK]단추를 찰칵한다.
4. 일람에 표시되여있는 [JBoss-IDE]를 선택하고 [Next]단추를 찰칵하면 지정한 URL로부터 설치가능한 JBoss-IDE의 검색을 시작한다.
5. 설치가능한 특성정보과 판본이 현시된다. 설치할 JBoss-IDE(여기서는 1.3.30)을 선택하고 [Next]단추를 찰칵한다.
6. 다음으로 허가의 확인화면, 설치할 특성정보의 확인화면이 련이어 나타나는데 [I Accept the agreement]를 선택하고 [Next]→[Finish]→[Install]의 순서로 진행한다.

7. 설치를 끝내면 Eclipse의 재기동을 독촉하는 대화칸이 나타나므로 재기동을 한다. 이것으로써 설치는 끝난다.

대리봉사기를 경유하여 JBoss-IDE를 설치하는 경우 [Window]→[Preferences(설정)]→Install/Update([설치/갱신)]을 선택하고 [대리설정]의 [Enable HTTP proxy connection]를 선택하여 대리봉사기의 주소와 포구를 입력한다.

11.2.3. JBoss-IDE의 설정

JBoss-IDE에서 JBoss봉사기의 기동, 정지 등을 진행하기 위한 설정을 한다.

1. [Window]→[Show View]→[Other]→[JBoss-IDE]→[Server Navigator]를 선택하고 [OK]단추를 찰칵하면 [Server Navigator]보기가 나타난다(그림 11-3).



그림 11-3. [Server Navigator] 보기

2. 다음으로 이번에 사용하는 JBoss봉사기의 구성에 관한 설정을 진행한다.

[Server Navigator]보기의 [configuration]단추를 찰칵하면 [Debug]대화칸이 나타난다(이 보기의 임의의 장소에서 오른쪽찰칵하여 [configuration]를 선택해도 된다). 여기서는 JBoss3.2.5를 전제로 하고있기때문에 화면왼쪽의 [configuration]안에서 [JBoss 3.2.x]를 오른쪽찰칵하고 [New]를 선택한다. 기정으로는 이 구성의 이름이 [New configuration]으로 되여있으므로 적당한 이름으로 변경한다. 여기서는 《JBoss3.2.5》라고 한다.

[Home]태브의 [JBoss 3.2.x Home Directory]에 이번에 리용하는 JBoss의 홈등록부를 입력한다. 여기서는 《C:/KUT/jboss-3.2.5》라고 입력한다. 다음으로 [Start]태브를 찰칵하고 [Program parameters(프로그람인수)]에 《-c all》를 입력한다. [JDK]태브에는 JRE가 아니라 JDK의 홈등록부를 지정한다. [Apply]→[Close]의 순서로 선택하면 설정이 끝난다(그림 11-4).

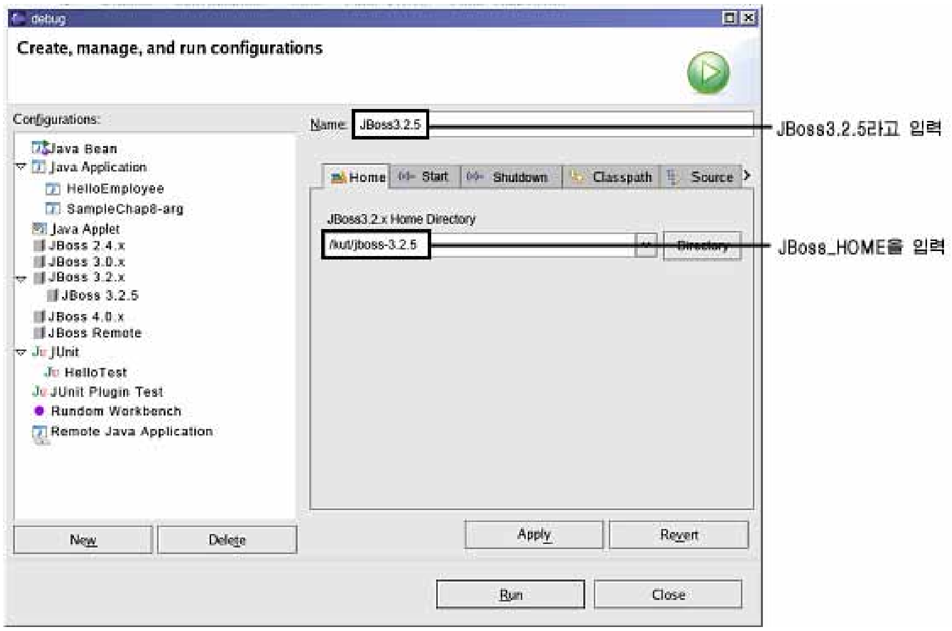


그림 11-4. JBoss구성화면

우의 설정에서 프로그람인수로 《-c all》를 입력하였는데 JBoss에는 [minimal], [default], [all]의 3가지 설정이 있다. 여기서는 EJB를 사용하기때문에 [all]설정을 사용한다. 이것을 위한 기동선택이 《-c all》이다. 자세한것은 표 11-2에 보여주었다.

표 11-2. JBoss봉사기표준설정값

설정값이름	개 요
minimal	JMX/JBoss만을 사용할 때 지정한다. Web용기, EJB용기, JMS는 포함하지 않는다.
Default	독립J2EE봉사기를 실행하기 위한 설정이다. RMI/IIOP와 Web봉사는 포함하지 않는다. 파라메터를 주지 않고 기정으로 JBoss봉사기가 기동한다.
All	리용가능한 봉사를 모두 개시한다. 여기에는 기정에서 적재하지 않는 RMI/IIOP, Web봉사를 포함하고있다.

Eclipse상에서 JBoss를 기동해보면 다음과 같다.

JBoss봉사기의 구성이 작성되면 [Server Navigator]보기에 [JBoss 3.2.5]가 표시된다. 그것을 선택하고 [start]단추를 찰칵하면 JBoss봉사기가 기동된다. 표준출력은 Eclipse상의 조종탁보기에 출력된다. 또한 JBoss봉사기의 정지는 [shutdown]단추를 찰칵하면 된다.

제3절. EJB의 작성과 실행

11.3.1. EJB응용쏘프트웨어의 작성

EJB(Enterprise JavaBeans)란 J2EE가동기반의 봉사기측요소이다. EJB에는 StatelessSessionBean, StatefullSessionBean, BMPEntityBean, CMPEntityBean, MessageDrivenBean이 있다. 여기서는 EJB에 대하여 구체적으로 설명하지 않고 JBoss-IDE를 사용한 EJB의 작성 및 실행방법에 대하여 설명한다.

이 장에서는 10장에서 작성한 종업원검색을 진행하는 응용프로그람을 EJB (StatelessSessionBean)를 사용하여 작성하기로 한다. 제10장에서는 JSP, Servlet 및 검색론리를 가지는 Java클라스로 처리를 했지만 여기서는 검색론리를 EJB로 바꾼다(그림 11-5).

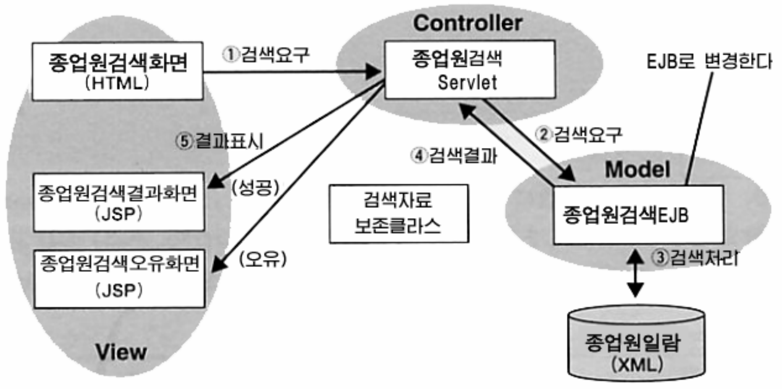


그림 11-5. EJB로 작성한 종업원검색응용프로그람에서의 처리방식

표 11-3은 종업원검색응용프로그람 EJB판의 각 구성요소의 개요와 제10장에서 변경된 내용을 보여준다. 화면변화에 대해서는 변경이 없으므로 제 10장과 같다.

표 11-3. 종업원검색응용프로그람 EJB판의 각 구성요소의 변경된 내용

명 칭	형 식	변 경 점
종업원검색화면	HTML	POST목적지의 URL를 변경한다.
종업원검색Servlet	Servlet	종업원검색클라스의 호출을 종업원검색EJB의 호출로 변경한다.
종업원EJB	EJB	Java클라스로부터 EJB로 변경한다. 자료검색의 처리자체에 변경은 없으나 메쏘드의 서명을 변경한다.
검색자료보존클라스	Java클라스	오유처리의 마당, 메쏘드를 추가한다.
종업원검색결과화면	JSP	자료보유클라스명을 변경한다.
종업원검색오유화면	JSP	자료보유클라스명을 변경한다.

11.3.2. 프로젝트의 작성

종업원검색응용프로그람EJB판의 개발을 위하여 프로젝트를 작성한다.

여기서는 JBoss-IDE를 리용하여 J2EE프로젝트를 작성한다. Eclipse의 차림표띠에서 [File]→[New]→[Other]를 선택하고 [JBoss-IDE]→[J2EE Projects]→[J2EE1.3 Project]를 선택한 다음 [Next]단추를 찰칵한다. 프로젝트명으로 《SampleChap11》를 입력하고 [Next]단추를 찰칵한다(그림 11-6).

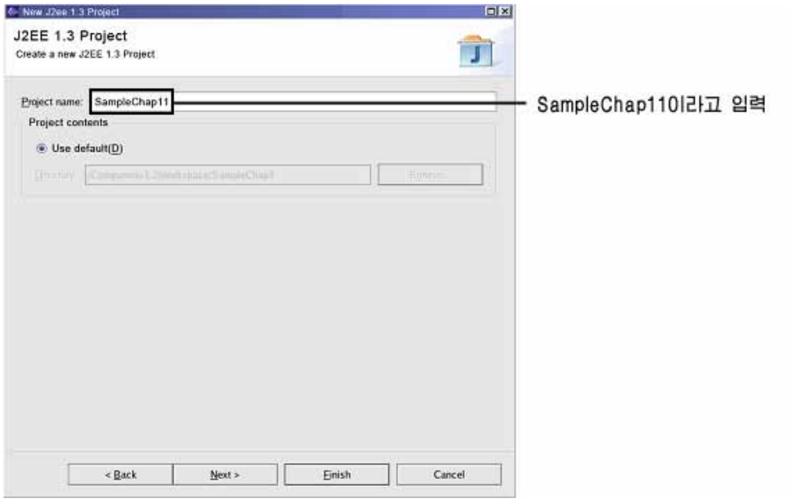


그림 11-6. J2EE프로젝트작성대화칸

다음으로 [Source Folder on the Build Path]과 [Default Output Folder]을 설정한다. [Add of Folder]를 누르면 [New Source Folder]대화칸이 나타난다. [Folder Name]에 《src》라고 입력하고 [OK]단추를 찰칵한다(그림 11-7).

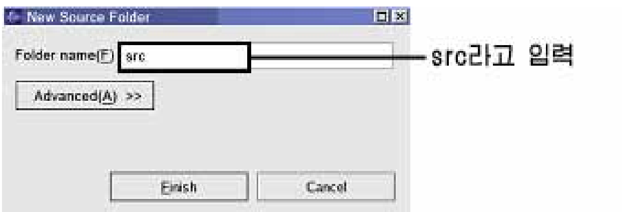


그림 11-7. 새 원천서류철이름입력대화칸

다음으로 [Default Output Folder]에 《SampleChap11/bin》을 입력하고 [Finish]단추를 찰칵한다(그림11-8).

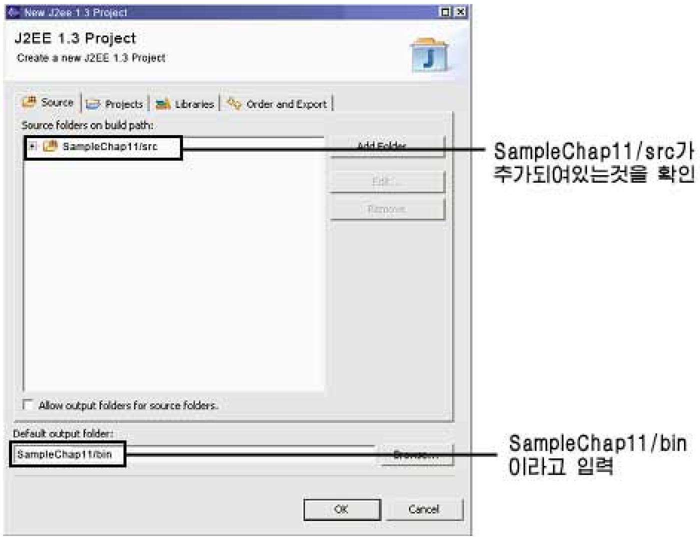


그림 11-8. J2EE프로젝트작성조수

이상으로 프로젝트작성은 끝난다.
패키지열람기상에서 SampleChap11가 등록되였는가를 확인한다.

11.3.3. 종업원검색화면(HTML)의 수정

제10장에서 작성한 종업원검색응용프로그람에 기초하여 EJB응용프로그람을 작성하기로 한다.

우선 종업원검색화면(index.html)을 작성한다. HTML이나 JSP를 넣는 서류철을 작성한다. 패키지열람기에서 [SampleChap11]을 오른쪽찰칵하고 [New]→[Folder]을 선택한다. [Folder name]에 《docroot》를 입력하고 [Finish]단추를 찰칵한다(그림11-9).

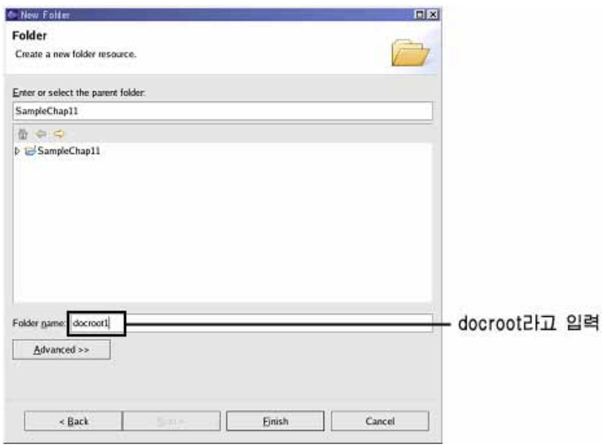


그림 11-9. 새 서류철작성대화칸

다음으로 이제 작성한 [docroot]를 오른쪽찰칵하고 [New]→ [Other]를 선택한다. [JBoss-IDE]→[Web Comments]→[HTML Page]를 선택하고 [Next]단추를 찰칵한다. [Input or select parent folder]에서 HTML를 보관할 서류철을 지정한다. 여기서는 방금 작성한 《docroot》를 지정한다. 직접 《SampleChap11/docroot》라고 입력하거나 화면중심에 있는 계층구조에서 《docroot》를 선택한다. 또한 화면아래의 [File name]에 《index.html》를 입력하고 [Finish]단추를 찰칵한다(그림 11-10).

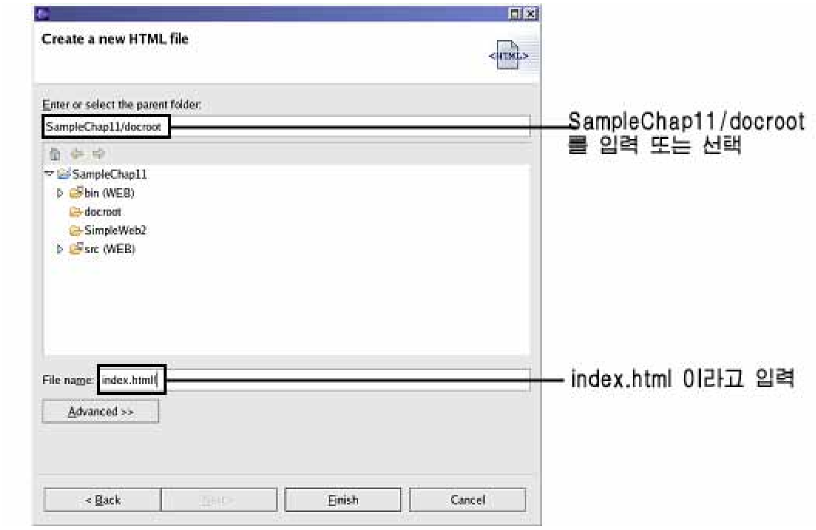


그림 11-10. 새로운 HTML작성대화칸

패키지열람기상에서 《index.html》이 작성되였는가를 확인한다. 《index.html》을 오른쪽찰칵하고 [Open With]→[Text Editor]를 선택하면 편집기상에 원천이 나타난다. 목록 11-1과 같은 코드를 입력한다.(굵은체로 되여있는 부분이 제10장의 코드에서 변경된 부분이다.) 제10장에서 작성한《index.html》와는 Servlet의 호출자리가 달라졌다.

목록 11-1. 종업원검색화면(index.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
    EN" "http://www.w3.org/TR/xhtml1-transitional.dtd">
<meta http-eqiv= "Content-Type" content= "text/html; charset=Shift_JIS">
<html>
  <head>
    <title>종업원검색</title>
  </head>
```

```
<body bgcolor= “#FFFFFF” >
<div align= “center” >
  <h3>검색하려는 종업원번호를 입력하십시오.</h3>

  <form name= “MemberSearchForm”  method= “post”
       action= “/SimpleWeb2/MemberSearchServlet2” >
    <table with= “300”  cellpadding= 10 align= “center” >
      <tr>
        <td width= “40%”  bgcolor= “lightblue” >
          <div align= “center” 종업원번호</div>
        </td>
        <td width= “60%”  bgcolor= “palegoldenrod” >
          <input type= “text”  name= “memberId” >
        </td>
      </tr>
    </table>
    <br>
      <input type= “submit”  name= “submit”  value= “검색실행” >
  </form>
</div>
</body>
</html>
```

11.3.4. 검색자료보존클라스의 작성

검색자료보존클라스(MemberDataHolder2.java)를 작성한다. 제10장의 종업원검색응용프로그람에서는 검색자료보존클라스의 구체례생성은 Servlet로 진행하고 참조넘기기로 종업원검색클라스에 넘겼다. EJB응용에서는 참조넘기기는 피하는것이 좋다. EJB에 원격접근한다는것은 인수의 객체는 직렬화되여 EJB에 넘겨진다는것을 의미한다. 즉 호출하는 측에서 생성된 검색자료보존클라스의 구체례를 EJB로부터는 참조할수 없다. 따라서 호출되는 측(EJB)에서의 변경내용은 호출하는 측(Servlet)의 구체례에는 반영되지 않는다. 다만 EJB를 국부접근으로 호출할 때 응용봉사기의 설정에 의하여 참조넘기기가 가능할 때도 있다.

이 장의 실례에서는 원격접근할 때의 작성방법을 소개한다. 패키지열람기에서 [SampleChap11]를 오른쪽찰칵하고 [New]→[Class]를 선택한다. [Package]에 《sample.common》을 입력하고 [Name]에 《MemberDataHolder2》를 입력하며 [Interface]에 [java.io.Serializable]를 추가한다. [Which method stubs wold you like to create?]의 선택을 모두 해제하고 [Finish]단추를 찰칵한다(그림 11-11). [java.io.Serializable]을 실현하는 리유는 객체의 직렬화를 가능하게 하기 위해서이다.

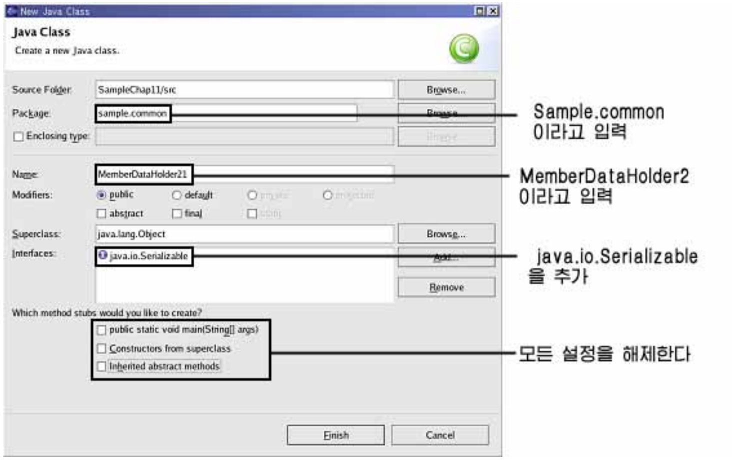


그림 11-11. 새로운 Java클라스작성대화칸

MemberDataHolder2.java가 작성되고 편집기에 표시되는데 목록 11-2의 코드를 입력한다(굵은체로 된 부분은 제10장의 코드에서 추가 또는 수정할 코드이다.). EJB에 오유가 있는가를 판정하기 위하여 boolean형의 변수와 메쏘드를 추가하고있다.

목록 11-2. 종업원자료보유클라스(MemberDataHolder2.java)

```
/*
 * 작성날자 200X/XX/XX
 *
 * TODO 이 생성된 파일의 형판을 변경하려면 다음으로 넘어뛰기:
 * 창문    설정    Java 코드격식   코드형판
 */
package sample.common;

import java.io.Serializable;

/**
 * @author KUT
 *
 * TODO이 생성된 형해설문의 형판을 변경하려면 다음으로 넘어뛰기:
 * 창문    설정    Java    코드격식    코드형판
 */

Public class MemberDataHolder2 implements Serializable {
   private String id;
   private String name;
   private String address;
```

```
private String message;
private Boolean isError;

/**
 * @return address를 되돌린다.
 */
public String getAddress() {
    return address;
}
/**
 * @param address address를 설정
 */
public void setAdress(String address) {
   this.address = address;
}
/**
 * @return id 를 돌린다.
 */
public String getId() {
   return id;
}
/**
 * @param id id를 설정.
 */
public void setId(String id) {
   this.id = id;
}
/**
 * @return isError를 돌린다.
 */
public Boolean isError() {
   return isError;
}
/**
 * @param isError isError를 설정.
 */
public void setError(Boolean isError){
  this.isError = isError;
}
/**
 * @return message를 돌린다.
 */
```

```
    public String getMessage() {
        return message;
    }
    /**
     * @param message message를 설정.
     */
    public void setMessage() {
        this.message = message;
    }
    /**
     * @return name 를 돌린다.
     */
    public String getName() {
        return name;
    }
    /**
     * @param name name을 설정.
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

11.3.5. 종업원검색EJB의 작성

종업원검색EJB(MemberSearchBean.java)를 작성한다. 여기서는 StatelessSessionBean을 작성하지만 다른 EJB(EntityBean이나 MessageDrivenBean)도 쉽게 작성할수 있다. [SampleChap11]를 오른쪽찰칵하고[New]→[Other]를 선택한다. [JBoss-IDE]→[EJB Components]→[Session Bean]을 선택하고 [Next]단추를 찰칵한다.

[Package]에는 《sample.ejb》, [Name]에는 《MemberSearchBean》를 입력한다. [Interface]에는 《javax.ejb.SessionBean》, [Select the type of the EJB]에서는 [Stateless]를 선택하고 [Select the access of the EJB]에서는 [Remote]를 선택한다. [Which method stubs would you like to create?]에서는 모두 선택하고 [Finish]단추를 찰칵한다(그림 11-12).

다음으로 EJB로서 외부에 공개하는 메쏘드를 추가한다.

패키지열람기에서 [MemberSearchBean]을 오른쪽찰칵하고 [J2EE]→[As Business Method]를 선택한다. 여기서 [MemberSearchBean.java](파일)과 [MemberSearchBean](클라스)의 차이에 주의해야 한다. [MemberSearchBean.java]를 오른쪽찰칵했다면 차림표에 [J2EE]는 나타나지 않는다.

[Method Name]에 《search》를 입력하고 [Return Type]에 앞에서 작성한 《sample.common.MemberDataHolder2》를 입력한다. [Parameters]에 java.lang.String

형의 《id》를 추가하고 [Select the view type to use]에서는 [Remote]를 선택한 다음 [Finish]단추를 찰칵한다(그림 11-13). 이렇게 하여 외부공개메쏘드가 추가된다.

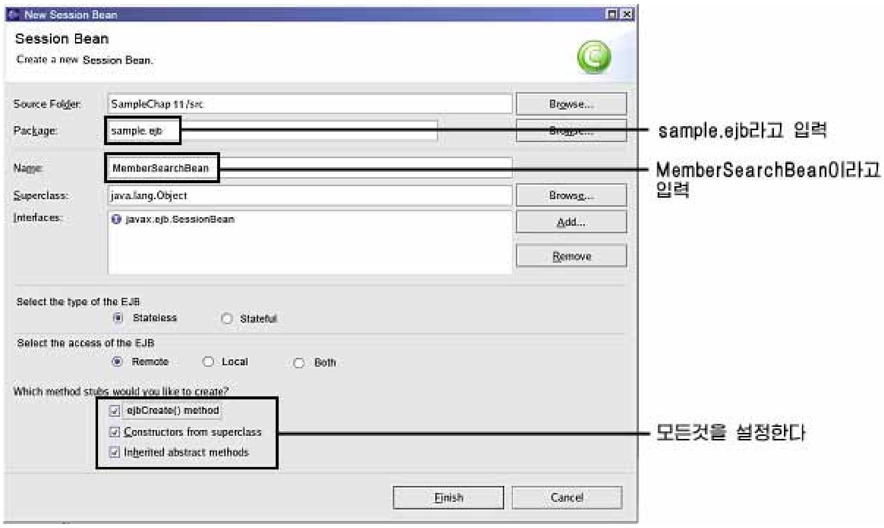


그림 11-12. SessionBean작성대화칸

편집기로 직접 입력하는 방식과 다른 점은 XDoclet와 관련된 점이다. 추가한 메쏘드의 해설문에 [@ejb]로 시작되는것이 추가되였다. 해설문을 자동적으로 추가하고있다. 목록 11-3의 search()메쏘드의 JavaDoc해설문을 참조하면 [@ejb.interface-method view-type= "remote"]가 자동삽입되여있음을 알수 있다.

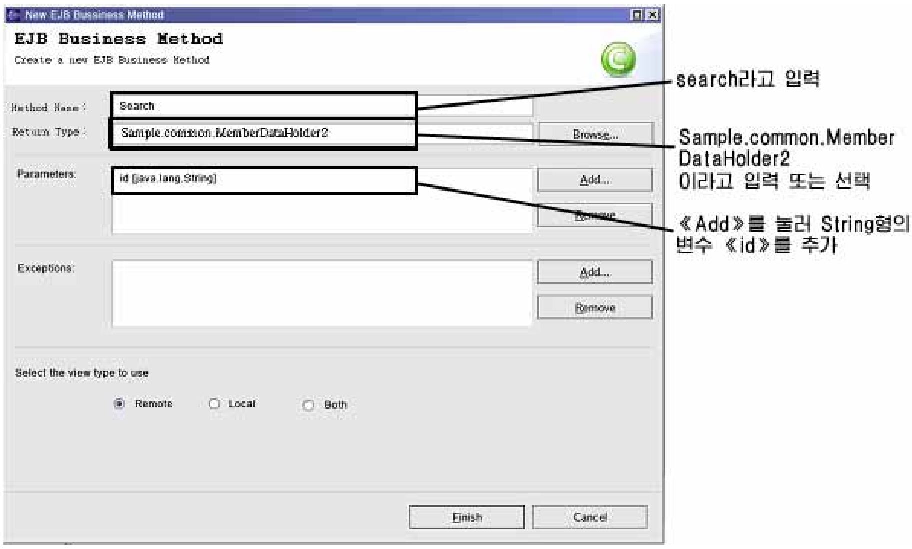


그림 11-13. 업무메쏘드작성대화칸

목록 11-3과 같이 코드를 입력한다. Search()메쏘드의 내용은 제10장에서 작성한 MemberSearch클라스와 거의 같다.

목록 11-3. 종업원검색EJB(MemberSearchBean.java)

```
/*
 * 작성날자: 200X/XX/XX
 *
 * TODO가 생성된 파일의 형판을 변경하려면 다음으로 건너뛰기:
 * 창문  설정  Java  코드격식  코드형판
 */
package sample.ejb;

import java.io.File;
import java.rmi.RemoteException;

import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.ModeList;

import sample.common.MemberDataHolder2;

/**
 * @ejb.bean name= "MemberSearch"
 *             display-name= "Name for MemberSearch"
 *             description= "Description for MemberSearch"
 *             jndi-name= "ejb/MemberSearch"
 *             type= "Stateless"
 *             view-type= "remote"
 */

public class MemberSearchBean implements SessionBean {
   private static final String filePath = "C://ASCII//workspace
             /SampleChap11//SimpleWeb2//member.xml" ;
   private static final String msgInputError =
             "종업원ID가 입력되지 않았습니다" ;
```

```
private static final String msgSearchSuccess =
            "종업원정보검색에 성공하였습니다.";
private static final String msgSearchError =
             "해당되는 종업원정보가 없었습니다.";
private static final String msgFailure  =
             "종업원정보의 검색에서 장애가 발생하였습니다.";

private static ModeList memberList;

private SessionContext ctx;

/**
 *
 */

public MemberSearchBean() {
   super();
   // TODO 자동생성된 메쏘드대용체
}

/* (Javadoc가 아님)
 * @see javax.ejb.SessionBean#ejbActivate()
 */
public void ejbActivate() throws EJBException, RemoteException {
   // TODO 자동생성된 메쏘드대용체
}

/* (Javadoc가 아님)
 * @see javax.ejb.Session#ejbPassivate()
 */
public void ejbPassivate() throws EJBException,RemoteException {
   // TODO 자동생성된 메쏘드대용체
}

/* (Javadoc가 아님)
 * @see javax.ejb.SessionBean#ejbRemove()
 */
public void ejbRemove() throws EJBException, RemoteException {
   // TODO 자동생성된 메쏘드대용체
}

/* (Javadoc가 아님)
 * @see javax.ejb.SessionBean#setSessionContext(javax.ejb.SessionConte
 * xt)
 */
```

```
public void setSessionContext(SessionContext ctx)
        throws EJBException, RemoteException {
   This.ctx = ctx;
}

/**
 * Default create method
 *
 * @throws CreateException
 * @ejb.create-method
 */

public void ejbCreate() throws CreateException {
     //TODO auto-generated method stub
}

/**
 * Business method
 * ejb.interface-method view-type = "remote"
 */

public sample.common.MemberDataHolder2 search(java.lang.String id) {
    // 반환용검색자료보존클라스생성
   MemberDataHolder2 mdh = new MemberDataHolder2();

   Try {
      // memberList가 없는 경우 member.xml로부터 얻기
      if (memberList == null) {
         DocumentBuilderFactory dbfactory =
                DocumentBuilderFactory.newInstance();
         DocumentBuilder builder =
             dbfactory.newDocumentBuilder();
         Document doc = builder.parse(new File(filePath));
         Element root = doc.getDocumentElement();
         memberList = root.getElementByTagName( "member" );
      }

      // 종업원ID가 빈 문자인 경우
      if (id == null || id.equals( " " )) {
        mdh.setMessage(msgInputError);
        mdh.setError(true);
        return mdh;
      }
```

```
        // 종업원ID를 설정
        mdh.setId(id);

        // memberList로부터 넘겨 받은 종업원ID와 같은 종업원을 검색
        for (int i = 0; i < memberList.getList.getLength(); i++) {

            // 넘겨 받은 종업원ID인가.
            if (id.equals(element.getAttribute( “id” ))) {
                NodeList nameList = element.getElementByTagName( “name” );
                Element nameElement = (Element) nameList.item(0);
                mdh.setName(nameElement.getFirstChild().getNodeValue());

                NodeList addressList = Element.getElementByTagName( “address” );

                Element addressElement = (Element) addressList.item(0);
                mdh.setAddress(
                    addressElement.getFirstChild().getNodeValue());

                    // 찾으려는 종업원이 존재하는 경우
                mdh.setMessage(msgSearchSuccess);

                return mdh;
            }
        }

        // 찾으려는 종업원이 존재하지 않는 경우
        mdh.setMessage(msgSearchError);
        mdh.setError(true);
    }
    cath (Exception e) {
        // 례외가 발생한 경우
        mdh.setMessage(msgFailure);
        mdh.setError(true);
    }
    return mdh;
  }
}
```

다음으로 종업원목록(member.xml)을 작성하는데 내용은 제10장에서 작성한 member.xml와 같다. 다만 파일의 배치장소는 《SampleChap11/SimpleWeb2/member.xml》이다.

11.3.6. EJB와 관련한 파일의 작성

EJB와 관련하는 클라스(홈대면부, 원격대면부) 및 서술자(ejb-jar.xml, jboss.xml)는 Xdoclet가 자동생성한다. 여기서는 Xdoclet의 정의를 진행한다. [SampleChap11]를 오른쪽찰칵하고 [Properties]을 선택한다. [SampleChap11의 속성]대화칸에서 [Xdoclet Configurations]를 선택하면 화면오른쪽에 Xdoclet의 정의화면이 표시된다. 상부령역에서 오른쪽찰칵하고 [Add]단추를 선택하면 [Configuration Name]대화칸이 표시되기때문에 [Name]에《EJB》라고 입력하고 [OK]단추를 찰칵한다. 이것으로 EJB용의 새로운 설정이 작성되였다(그림 11-14).

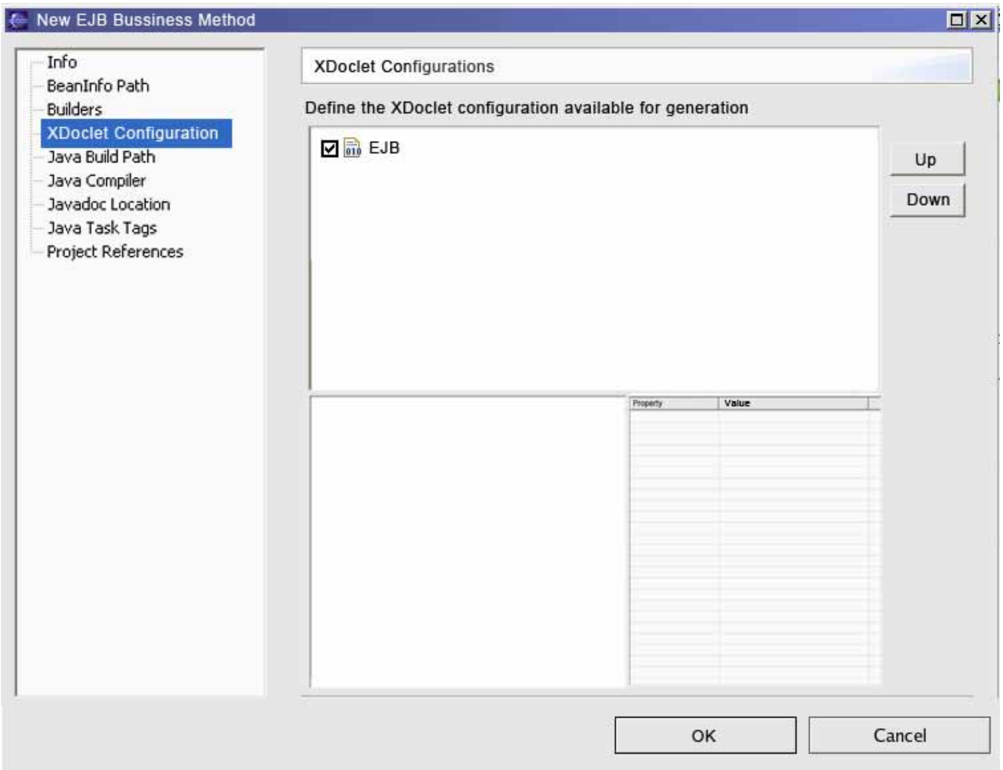


그림 11-14. Xdoclet정의대화칸

작성된 [EJB]는 상부령역에 표시되게 된다. 상부령역의 [EJB]를 선택하고 왼쪽아래령역에서 오른쪽찰칵하고 [Add Doclet]를 선택한다. 그러면 [Choose an entry]대화칸이 나타나는데 여기서 [ejbdoclet]를 선택하고 [OK]단추를 찰칵한다(그림 11-15).

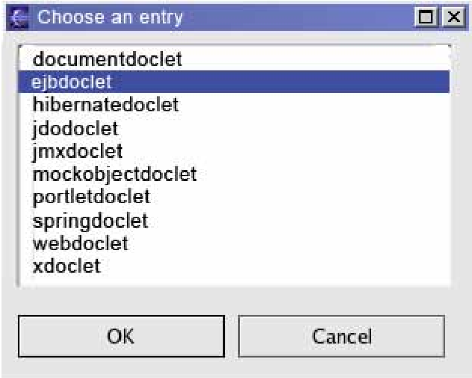


그림 11-15. [Choose an entry] 대화칸

왼쪽아래령역에 [ejbdoclet]가 있는것을 확인할수 있다. [destDir]를 두번찰칵하면 현시되는 [Attribute Value]대화칸에 《src》라고 입력하고 [OK]단추를 찰칵한 다음 검사칸에 검사표시를 한다. 마찬가지로 [ejbSpec]에 《2.0》라고 입력하고 검사표시를 한다. 여기서 검사표시를 하지 않으면 우에서 정의한 설정이 무효로 된다(그림 11-16).

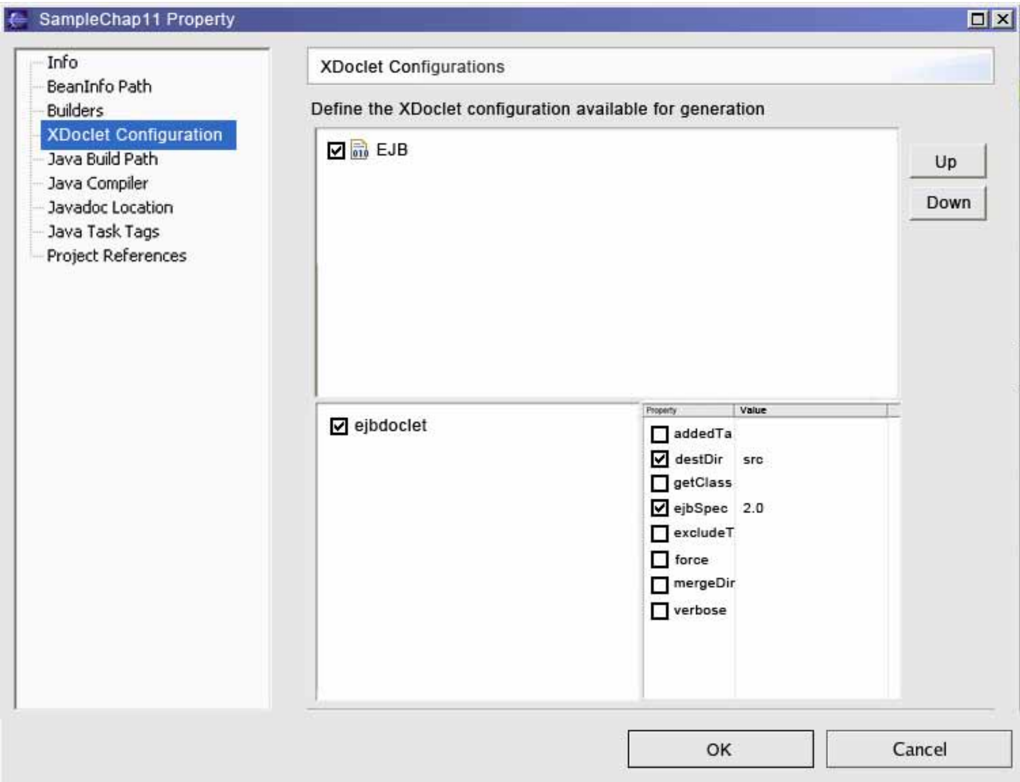


그림 11-16. ejbdoclet의 정의

다음으로 부과제를 추가한다. 왼쪽아래령역의 [ejbdoclet]를 오른쪽찰칵하고 [Add]단추를 선택한다. [Choose an entry]대화칸이 현시되는데 여기서 [fileset]를 선택하고 [OK]단추를 찰칵한다. 왼쪽아래령역에 [ejbdoclet]의 부과제로서 [fileset]가 나타난다. [fileset]를 선택하고 오른쪽아래령역의 [Property]에서 우와 같은 방법으로 입력한다.

[dir]에 《src》를 입력하고 검사표시하며 [includes]에 《**/*Bean.java》를 입력하고 검사표시한다. [excludes]는 검사표시를 해제한다. [includes]의 지정은 원천파일명의 마지막에 [Bean.java]라고 붙은것만 EJB로서 인식시키기 위한것이다(그림 11-17).

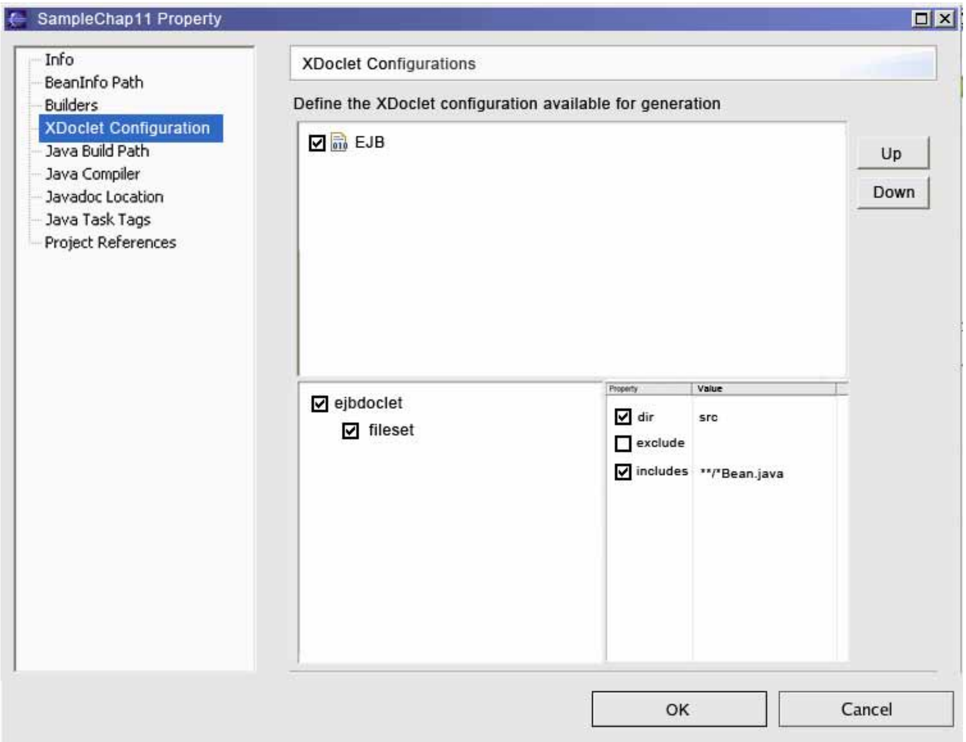


그림 11-17. fileset의 정의

이제부터 지금까지의 진행한 방법대로 정의를 진행한다.

[ejbdoclet]를 오른쪽찰칵→[Add]를 선택→목록으로부터 [deploymentdescriptor]를 선택→ [OK]단추를 찰칵한다. 왼쪽아래령역에서 [deploymentdescriptor]를 선택하고 오른쪽아래령역의 [dest Dir]에 《src/META-INF》를 입력하고 검사표시한다.

다음으로 [ejbdoclet]를 오른쪽찰칵→[Add]를 선택→목록으로부터 [jboss]를 선택→[OK]단추를 찰칵한다. 왼쪽아래령역에서 [jboss]를 선택하고 오른쪽아래령역에서 [Version]에 《3.0》을 입력하고 검사표시하며 [destDir]에 《src/MET-INF》를 입력

하고 검사표시한다.

또한 [ejbdoclet]를 오른쪽찰칵→[Add]를 선택→목록으로부터 [packageSubstitution]을 선택→[OK]를 찰칵한다. 왼쪽아래령역에서 [packageSubstitution]을 선택하고 오른쪽아래령역에서 [packages]에 《ejb》를 입력하고 검사표시하며 [substituteWith]에 《interfaces》를 입력하고 검사표시한다.

마지막으로 [ejbdoclet]를 오른쪽찰칵→[Add]를 선택→목록에서 [remoteinterface]를 선택→[OK]를 찰칵한다. 마찬가지로 [homeinterface]도 선택한다. 이 부과제들에 대하여서는 속성지정을 하지 않는다. 이상으로 설정은 끝난다(그림 11-18). [OK]단추를 찰칵하고 완료한다.

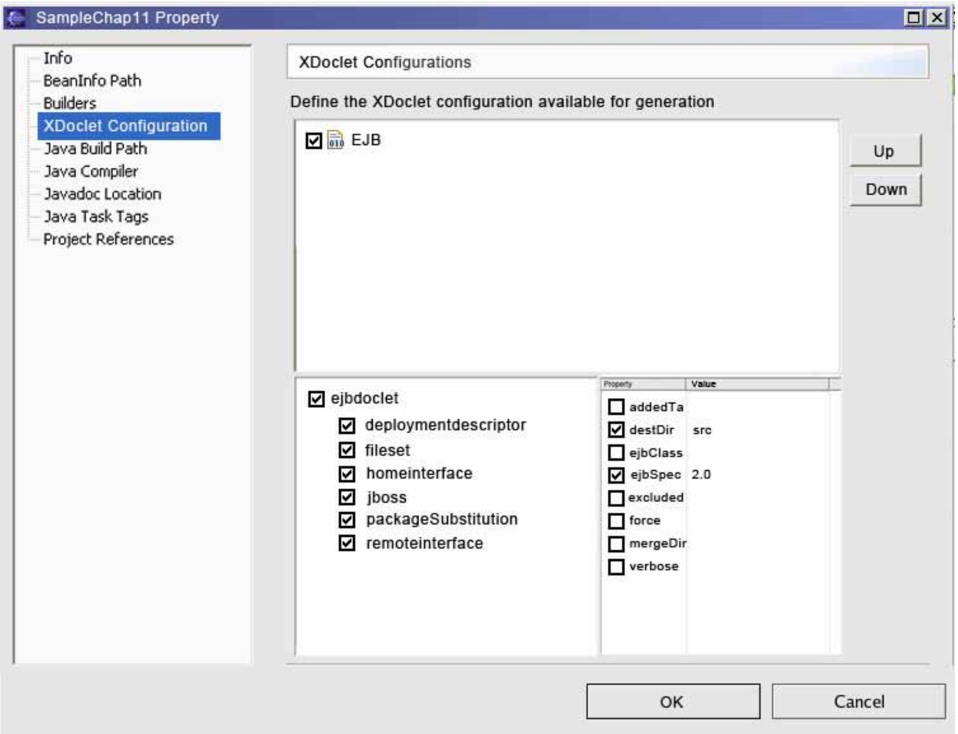


그림 11-18. 입력이 끝난 Xdoclet정의대화칸

이제 Xdoclet를 실행시켜본다. 패키지열람기상에서 [SampleChap11]를 오른쪽찰칵하고 [Run Xdoclet]를 선택한다. [SampleChap11/src/sample/interfaces/MemberSearch.java], [SampleChap11/src’sample/interfaces/MemberSearchHome.java], [SampleChap11/src/META-INF/ejb-jar.xml], [SampleChap11/src/META-INF/jboss.xml]가 작성된다. 패키지열람기상에서 확인할수 없는 경우에는 F5건을 누르면 화면이 갱신된다. 실행결과는 조종탁보기에 출력된다(그림 11-19).



그림 11-19. Xdoclet의 실행결과

출력된 파일의 내용은 목록 11-4～목록 11-7에 있다.

목록 11-4. 원격대면부(MemberSearch.java)

```
/*
 * Generated by Xdoclet   Donot edit!
 */
package sample.interfaces;

/**
 * Remote interface for MemberSearch.
 */
public interface MemberSearch extends javax.ejb.EJBObject {
   /**
    * Business method
    */
   public sample.common.MemberDataHolder2 search
      (java.lang.String id) throws java.rmi.RemoteException;
}
```

목록 11-5. 홈대면부(MemberSearchHome.java)

```
/*
 * Generated by Xdoclet    Do not edit!
 */
package samle.interfaces;

/**
 * Home interface for MemberSearch.
 */
public interface MemberSearchHome extends javax.ejb.EJBHome {
   public static final String COMP_NAME= "java:comp/env/ejb/MemberSearch";
   public static final String JNDI_NAME= "ejb/MemberSearch";
```

```
    public sample.interfaces.MemberSearch create()
        throws javax.ejb.CreateException,java.rmi.RemoteException;
}
```

목록 11-6. EJB서술자(ejb-jar.xml)

```
<?xml version= “1.0” encoding= “UTF-8” ?>
<!DOCTYPE ejb-jar PUBLIC “-//Sun Microsystems, Inc.
    //DTD Enterprise JavaBeans 2.0//EN” “http://java.sun.com/dtd/ejb-jar_2_0.dtd” >
<ejb-jar>
<description><![CDATA[No Description.]]></description>
<display-name>Generated by Xdoclet</display-name>
<enterprise-beans>
<!--Session Beans -->
<session>
   <description><![CDATA[Description for MemberSearch]]></description>
   <display-name>Name for MemberSearch</display-name>

   <ejb-name>MemberSearch</ejb-name>

   <home>sample.interfaces.MemberSearchHome</home>
   <remote>sample.interfaces.MemberSearch</remote>
   <ejb-class>sample.ejb.MemberSearchBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
</session>

<!--
   To add session beans that you have deployment descriptor info for,
      add a file to your Xdoclet merge directory called entity-beans.xml
      that contains the <entity></entity>markup for those beans.
-->

<!-- Entity Beans -->
<!--
  To add entity beans that you have deployment descriptor info for, add
  a file to your Xdoclet merge directory called entity-beans.xml that
  contains the <entity></entity> markup for those beans.
-->

<!-- Message Driven Beans -->
<!--
```

```
    To add message driven beans that you have deployment descriptor info
    for, add a file to your Xdoclet merge directory called message-driven-beans.xml
    that contains the <message-driven></message-driven> markup for those beans.
-->

</enterprise-beans>

<!-- Relationships -->

<!-- Assembly Descriptor -->
<assembly-descriptor >
<!--
    To add additional assembly descriptor info here, add a file to your
    Xdoclet merge directory called assembly-descriptor.xml that contains
    the assembly-descriptor></assembly-descriptor> markup.
-->

<!-- finder permissions -->

<!-- transactions -->

<!-- finder transactions -->
</assembly-descriptor>

</ejb-jar>
```

목록 11-7. EJB서술자(jboss.xml)

```
<?xml version= "1.0"  encoding= "UTF-8" ?>
<!DOCTYPE jboss PUBLIC  "-//Jboss//DTD JBOSS 3.0//EN"
        "http://www.jboss.org/j2ee/dtd/jboss_3_0.dtd" >
<jboss>
   <enterprise-beans>
      <!--
      To add beans that you have deployment descriptor info
      for, add a file to yor Xdoclet merge directory called jboss-bean
      s.xml that contains the <session></session>, <entity></entity> a
      nd <message-driven></message-driven> markup for beans.
       -->

      <session>
         <ejb-name>MemberSearch</ejb-name>
```

```
            <jndi-name>ejb/MemberSearch</jndi-name>
        </session>
    </enterprise-beans>

    <resource-managers>
    </resource-managers>
</jboss>
```

11.3.7. 종업원검색Servlet의 작성

종업원검색Servlet(MemberSearchServlet2.java)를 작성한다.

[SampleChap11]를 오른쪽찰칵하고 [New]→[Other]를 선택한다. [Jboss-IDE]→[Web Components]→[HTTP Servlet]를 선택하고 [Next]단추를 찰칵한다.

[Package]에 《sample.web》를 입력하고 [Name]에 《MemberSearchServlet2》를 입력한다. [Modifiers]에 [public]가 선택되여있는가를 확인하고 [Super class]에 《javax.servlet.http.HttpServlet》가 입력되여있는가를 확인한다. [Which method stubs would like to create?]에서는 [init()method]를 선택하고 [Which service method stubs would like to create?]에서는 [doPost()method]를 선택한 다음 [Finish]단추를 찰칵한다(그림 11-20).



그림 11-20. Servlet작성대화칸

[MemberSearchServlet2.java]가 작성되여 편집기에 표시되므로 목록 11-8의 코드를 입력한다(굵은체로 되여있는 부분이 제10장의 코드에 대한 추가 또는 수정할 코드이다.). 종업원검색Servlet에서는 초기처리로서 종업원검색EJB의 홈대면부를 얻고 보유한다. 실행메소드(doPost())에서는 요청(request)으로부터 얻은 종업원번호(memberId)에 근거하

여 종업원검색EJB(MemberSearchBean)에 검색을 의뢰하며 검색이 성공한 경우(되돌이값인 MemberDataHolder2의 isError()메쏘드가 허위) 종업원검색결과화면(/memberSearchOK.jsp)을 현시하고 오유가 발생한 경우(MemberDataHolder2의 isError()메쏘드가 참) 또는 례외가 발생한 경우 종업원검색오유화면(/memberSearchNG.jsp)을 현시하도록 하는 처리를 진행하고있다.

목록 11-8. 종업원검색Servlet(MemberSearchServlet2.java)

```
/*
 * 작성날자: 200X/XX/XX
 *
 * Todo가 생성된 파일의 형판을 변경시키려면 다음으로 건너뛰기:
 * 창문    설정    Java    코드격식    코드형판
 */
package sample.web;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.servlet.http.HttpServlet;

import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import java.io.IOException;
import java.util.Hashtable;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import sample.common.MemberDataHolder2;
import sample.interfaces.MemberSearch;
import sample.interfaces.MemberSearchHome;

/**
 * Servlet Class
 *
 * @web.servlet name= "MemberSearchServlet2"
 * display-name= "Name for MemberSearchSevlet2"
 * @web.servlet-mapping  url-pattern= "MemberSearchServlet2"
 * @web.ejb-ref name= "ejb/MemberSearch"
```

```
 * type= "Stateless"
 * home= "sample.interfaces. MemberSearchHome"
 * remote= "sample.interfaces.MemberSearch"
 * description= "Reference to the MemberSearch EJB"
 * @jboss.ejb-ref-jndi  ref-name= "ejb/MemberSearch"
 * jndi-name= "ejb/MemberSearch"
 */
public class MemberSearchServlet2 extends HttpServlet {

   private static final String ngPage =  "/memberSearchNG.jsp" ;
   private static final String okPage =  "/memberSearchOK.jsp" ;

   private MemberSearchHome home;

   public void init(ServletConfig config) throws ServletException {
      super.init(config);
      try {

         Hashtable props = new Hashtable();
         props.put(InitialContext.INITIAL_CONTEXT_FACTORY,
            "org.jnp.interfaces.NamingContextFactory" );
         pros.put(InitialContext.PROVIDER_URL,  "jnp://127.0.0.1:1099" );
         Context context = new InitialContext(props);

         Object ref = context.lookup(MemberSearchHome.COMP_NAME);
         This.home = (MemberSearchHome) PortableRemoteObject.
         Narrow(ref, MemberSearchHome.class);
      }
      catch (Exception e) {
         throw new ServletException(e);
      }
   }

   protected void doPost(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {

      // 이행측JSP
      String resultPage;
      try {
         // 종업원ID얻기
         String memberId = request.getParameter( "memberId" );
         // EJB호출
```

```
            MemberSearch remote = this.home.create();
            MemberDataHolder2 mdh = remote.search(memberId);

            // 결과판정
            if (!mdh.isError()) {
               resultPage = okPage;
            } else {
               resultPage = ngPage;
            }

            // 화면호출
            request.setAttribute( "mdh" , mdh);
            javax.servlet.RequestDispatcher dispatch =
                request.getRequestDispatcher(requestPage);
            dispatch.forward(request, response);
         }
         catch (Exception e) {
            javax.servlet.RequestDispatcher dispatch =
                request.getRequestDispatcher(ngPage);
            dispatch.forward(request, response);
         }
      }
   }
```

11.3.8. Servlet와 관련된 파일의 생성

Servlet와 관련된 파일(web.xml, jboss-web.xml)은 XDoclet를 생성한다. 여기서는 XDoclet의 정의를 진행한다. [11.3.6. EJB와 관련된 파일의 작성]과 절차상 차이는 없다.

패키지열람기로 [SampleChap11]을 오른쪽찰칵하고 [Properties]을 선택한다. [SampleChap11의 속성]에서 [XDoclet configurations]를 선택하면 화면오른쪽에 XDoclet의 정의화면이 나타난다. 상부령역에서 오른쪽찰칵하고 [Add]단추를 선택하면 [Configuration Name]대화칸이 나타나는데 [Name]에 《Web》라고 입력하고 [OK]단추를 찰칵한다. 이것으로 웨브용의 새로운 설정이 작성된다.

다음으로 작성된 [Web]를 선택하고 왼쪽아래령역에서 오른쪽찰칵하여 [Add Doclet]를 선택한다. [Choose an entry]대화칸이 나타나는데 [webdoclet]를 선택하고 [OK]단추를 찰칵한다. 왼쪽아래령역에 [webdoclet]가 나타나는것을 확인할수 있다. [webdoclet]를 선택하면 오른쪽령역에 [Property]가 현시된다. [destDir]를 두번찰칵하여 현시되는 [Attribute Value]대화칸에서 《src/WEB-INF》라고 입력하고 [OK]단

추를 찰칵한다.

다음으로 부과제를 추가한다. 왼쪽아래령역의 [webdoclet]를 오른쪽찰칵하고 [Add]단추를 선택한다. [Choose an entry]대화칸이 표시되므로 [fileset]를 선택하고 [OK]단추를 찰칵한다. 왼쪽아래령역에 [webdoclet]의 부과제로서 [fileset]가 나타난다. [fileset]를 선택하고 오른쪽령역의 [dir]에 《src》를 입력하고 선택하며 [includes]에 《**/*Servlet2.java》를 입력하고 선택한다. [excludes]의 선택은 해제한다. 여기서 [includes]지정은 원천파일명의 마지막에 《Servlet2.java》이 붙은것만을 Servlet로 인식시키기 위한것이다.

이제부터의 작업은 지금까지의 작업의 반복으로 된다. [webdoclet]를 오른쪽찰칵→[Add]를 선택→목록에서 [deploymentdescriptor]를 선택 →[OK]를 찰칵한다. 왼쪽아래령역에서 [deploymentdescriptor]를 선택하고 오른쪽아래령역에서 [Servletspec]에 《2.3》을 입력하며 선택한다. 여기서 [destDir]에 입력하지 않는 리유는 [webdoclet]에서 지정한 [destDir]가 계승되기때문이다. 부과제는 상위과제의 속성을 이어받는다.

끝으로 [webdoclet]를 오른쪽찰칵→[Add]를 선택→목록에서 [jbosswebxml]를 선택→[OK]를 찰칵한다. 왼쪽아래령역에서 [jbosswebxml]을 선택하고 오른쪽아래령역의 [Version]에 《3.0》을 입력하며 선택한다. 여기서도 마찬가지로 [destDir]은 상위과제인 [webdoclet]부터 계승된다.

이상으로 설정이 끝는다. [OK]단추를 찰칵하여 완료한다(그림 11-21).

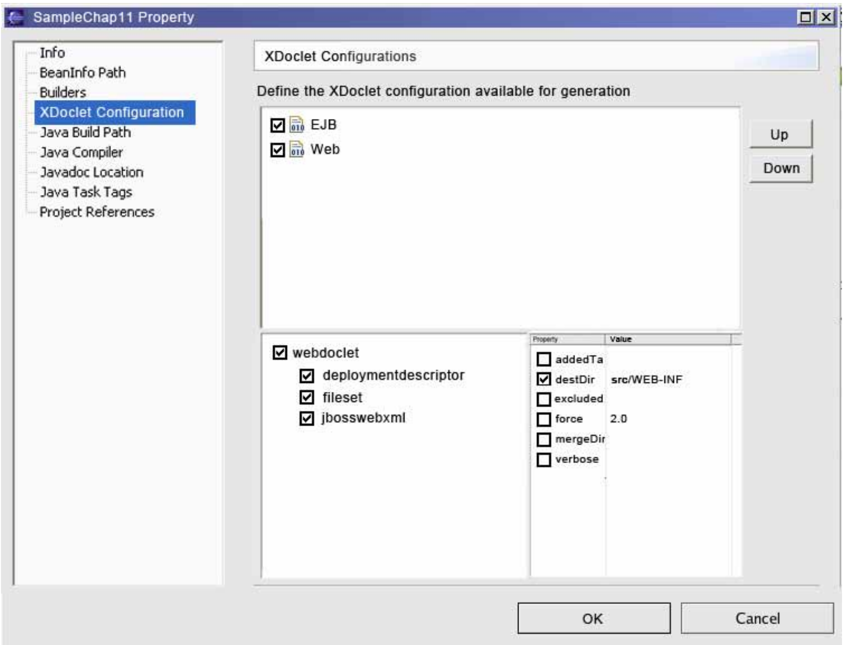


그림 11-21. 입력이 완성된 Xdoclet정의대화칸

이제 Xdoclet를 실행해본다. 패키지열람기상에서 [SampleChap11]를 오른쪽찰칵하고 [Run Xdoclet]를 선택한다.

[SampleChap11/src' WEB-INF/web.xml], [SampleChap11' src/ WEB-INF/ jboss-web.xml]이 생성된다. 패키지열람기상에서 작성된 파일을 확인할수 없을 때에는 F5건을 누른다. 출력된 파일의 내용을 목록 11-9～목록 11-10에 표시한다.

목록 11-9. Web서술자(web.xml)

```
<?xml version= “1.0” encoding= “UTF-8” ?>
<!DOCTYPE web-app PUBLIC “-//Sun Microsystems, Inc.
   //DTD Web Application 2.3//EN” “http://java.sun.com/dtd/web-app_2_3.dtd” >

<web-app>
  <distributable/>
    <!--
    To use non Xdoclet filters, create a filters.xml file that
    Contains the additional filters (eg Sitemesh) and place it in your
    project' s merge dir. Don' t include filter-mappings in this file,
    Include them in a file called filter-mappings.xml and put that in
    the same directory.
    -->

    <!--
    To use non Xdoclet filter-mappings, create a filter-mappings.xml
    file that contains the additional filter-mappings and place it in your
    project' s merge dir.
    -->

    <!--
    To use non Xdoclet listeners, create a listeners.xml file that contains
    the additional listeners and place it in your
    project' s merge dir.
    -->

    <!--
    To use non Xdoclet servlets, create a servlets.xml file that
    contains the additional servlets (eg Struts) and place it in your
    Project' s merge dir. Don' t include servlet-mappings in this file,
    Include them in a file called servlet-mappings.xml and put that in
    the same directory.
    -->
```

```
<servlet>
  <servlet-name>MemberSearchServlet2</servlet-name>
  <display-name>Name for MemberSearchServlet2</display-name>
  <description><![CDATA[Description for MemberSearchServlet2]]>
  </description>
  servlet-class>sample.web.MemberSearchServlet2</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>MemberSearchServlet2</servlet-name>
  <url-pattern>/MemberSearchServlet2</url-pattern>
</servlet-mapping>

<!--
To specify mime mappings, crate a file named mime-mappings.xml, put
it in your project' s mergedir.
Organize mime-mappings.xml following this DTD slice:

<!ELEMENT mime-mapping (extension, mime-type)>
-->

<!--
To specify error pages, crate a file named error-pages.xml,
put it in your project' s mergedir.
Organize error-pages.xml following this DTD slice:

<!ELEMENT error-pages ((error-code | exception-type), location)>
-->

<!--
To add taglibs by xml, create a file called taglibs.xml and place
it In your merge dir.
-->

<!--
To set up security settings for your web app, create a file named
Web-security.xml, put it in your project' s mergedir.
Organize web-security.xml following this DTD slice:

<!ELEMENT security-constraint (display-name?,
    web-resource-collection+, auth-constraint?,
```

```
          user-data-constraint?)>
      <!ELEMENT web-resource-collection (web-resource-name,
              description?, url-pattern*, http-method*)>
      <!ELEMENT web-resource-name (#PCDATA)>
      <!ELEMENT url-pattern (#PCDATA)>
      <!ELEMENT http-method (#PCDATA)>
      <!ELEMENT user-data-constraint (description?, transport-guarantee)>

      <!ELEMENT transport-guarantee (#PCDATA)>

      <!ELEMENT login-config (auth-method?, realm-name?, form-login-config?)>

      <!ELEMENT auth-method (#PCDATA)>
      <!ELEMENT realm-name (#PCDATA)>
      <!ELEMENT form-login-config (form-login-page, form-error-page)>
      <!ELEMENT form-login-page (#PCDATA)>
      <!ELEMENT form-error-page (#PCDATA)>
      -->

      <ejb-ref>
        <description><![CDATA{Reference to the MemberSearch EJB}]>
        </description>
        <ejb-ref-name>ejb/MemberSearch</ejb-ref-name>
        <ejb-ref-type>Stateless</ejb-ref-type>
        <home>sample.interfaces.MemberSearchHome</home>
        <remote>sample.interfaces.MemberSearch</remote>
      </ejb-ref>
  </web-app>
```

목록 11-10. Web서술기(jboss-web.xml)

```
  <?xml version= "1.0"  encoding= "UTF-8" ?>
  <!DOCTYPE  jboss-web  PUBLIC  "-//Jboss/DTD  Web  Application
  2.3//EN"  "http://wwww.jboss.org/j2ee/dtd/jboss-web_3_2.dtd" >

  <jboss-web>
  <!-- Resource Environment References -->
  <!--
      For additional resource-env-ref tags add a merge file called
      Jbossweb-resource-env-ref.xml
    -->
```

```
<!-- Resource references -->
<!--
    For additional resource-ref tags add a merge file called
    Jbossweb-resource-ref.xml
-->

<!-- EJB References -->
<!--
    For additional ejb-ref tags add a merge file called jbossweb-ejb-ref.xml
-->
<ejb-ref>
   <ejb-ref-name>ejb/MemberSearch</ejb-ref-name>
   <jndi-name>ejb/MemberSearch</jndi-name>
</ejb-ref>
<!-- EJB Local References -->
</jboss-web>
```

11.3.9. 종업원검색결과화면(JSP)의 작성

종업원검색결과화면(memberSearchOK.jsp)을 작성한다.

패키지열람기상에서 [docroot]를 오른쪽찰칵하고 [New]→[Other]를 선택한 다음 [JBoss-IDE]→[Web Components]→[JSP Page]를 선택하며 [Next]단추를 찰칵한다. [Input or select parent folder]에 《SampleChap11/docroot》가 입력되였음을 확인하고 [File name]에 《memberSearchOK.jsp》를 입력한 다음 [Finish]단추를 찰칵한다(그림 11-22).

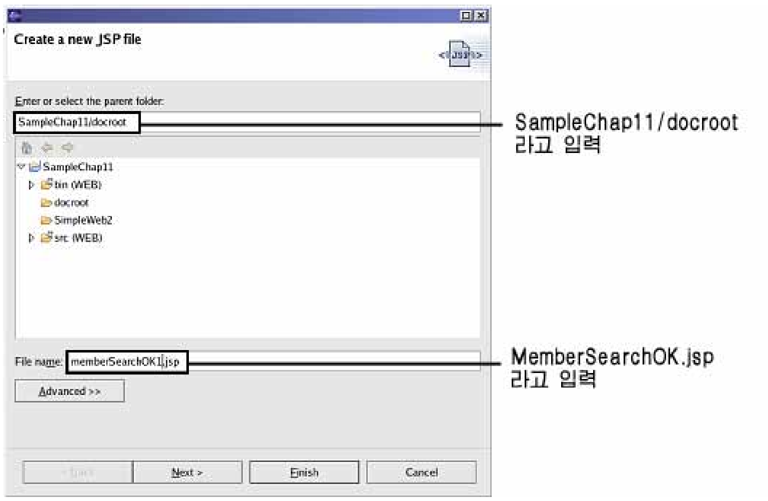


그림 11-22. JSP작성대화칸

목록 11-11과 같은 코드를 입력한다(굵은체로 된 부분이 제10장의 코드에서 추가 또는 수정한 코드이다.).

목록 11-11. 종업원검색결과화면(memberSearchOK.jsp)

```
<%@page language= “java”  contenttype= “text/html;charset=UTF-8”
     pageEncoding= “UTF-8”  %>
<!DOCTYPE html PUBLIC  “-//W3C//DTD XHTML 1.0 Transitional
  //EN”  “http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd” >
<html>
  <head>
    <title>종업원검색결과</title>
  </head>
  <body bgcolor= “#FFFFFF” >
    <jsp:useBean id= “mdh”  scope= “request”
        class= “sample.common.MemberDataHolder2”  />

      <div align= “center” >
        <h3>종업원검색결과</h3>
          <%= mdh.getMessage() %>

             <table width= “500”  cellpadding= 10 align= “center” >
              <tr>
                <td width= “20%”  bgcolor= “lightblue” >종업원번호</td>
                <td width= “80%” bgcolor= “palegoldenrod” align= “left” >
                  <%= mdh.getname() %></td>
              </tr>
              <tr>
                <td width= “20%”  bgcolor= “lightblue” >종업원명</td>
                <td width= “80%” bgcolor= “palegoldenrod” align= “left” >
                  <%= mdh.getName() %></td>
              </tr>
              <tr>
                <td width= “20%”  bgcolor=” lightblue” >주소</td>
                <td width=” 80%” bgcolor=” palegoldenrod” align=” left” >
                  <%= mdh.getAddress() %></td>
              </tr>
             </table>
      </div>
  </body>
</html>
```

11.3.10. 종업원검색오유화면(JSP)의 작성

종업원검색오유화면(memberSearchNG.jsp)을 작성한다.

종업원검색결과화면과 같은 절차로 작성한다. 출력서류철은 《SampleChap11/docroot》로 하고 파일명은 《memberSearchNG.jsp》라고 입력한다. 서류철이 작성된 다음 목록 11-12와 같이 코드를 입력한다. (굵은체로 된 부분이 제10장의 코드에서 추가 또는 수정할 코드이다.)

목록 11-12. 종업원검색오유화면(memberSearchNG.jsp)

```
<%@ page language=”java” contenttype=”text/html;charset=UTF-8”
       pageEncoding=”UTF-8” %>
<!DOCTYPE html PUBLIC “-//W3C/DTD XHTML 1.0 Transitional//EN”
  “http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html>
  <head>
    <title>종업원검색오유</title>
  </head>
    <body bgcolor=”#FFFFFF”>
      <jsp:useBean id=”mdh” scope=”request”
         class=”sample.common.MemberDataHolder2” />

         <div align=”center”>
           <h3>종업원검색오유</h3>
           <%= mdh.getMessage() %>
          </div>
     </body>
</html>
```

제4절. 패키지화

JBoss-IDE는 여러가지 보존파일의 패키지화를 간단하게 하는 방법을 제공하고있다. 패키지의 형식에는 제한이 없다.

이 절에서는 패키지화에 대하여 설명한다. 앞절에서는 실행에 필요한 파일을 작성하였는데 J2EE에서는 실행환경에 2진파일을 배치할 때 보존(*.jar, *war, *.ear)파일형식을 사용한다. 그러나 보존파일을 적재한 형식으로 배치할수도 있다. 이러한 방식은 정기적으로 갱신되는 정적파일을 포함하는 경우나 응용프로그람전체를 재배치하지 않고 부분적으로 갱신할 때에 리용된다.

11.4.1. application.xml의 작성

.ear파일에 필요하게 되는 [application.xml]을 작성한다. [src/META-INF]의 바로 아래에 파일을 작성하고 목록 11-13과 같이 코드를 입력한다.

목록 11-13. 기업급응용프로그람서술기(application.xml)

```
<?xml version= “1.0” encoding= “UTF-8” ?>
<!DOCTYPE application PUBLIC “-//Sun Microsystems, Inc.
   //DTD J2EE Application1.3//EN” “http://java.sun.com/dtd/application_1_3.dtd” >
<application>
  <display-name>SimpleApp</display-name>
  <module>
    <ejb>SimpleEJB.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>SimpleWeb2.war</web-uri>
      <context-root>/SimpleWeb2</context-root>
    </web>
  </module>
</application>
```

11.4.2. EJB요소의 패키지화

우선 EJB를 패키지화한다. 패키지열람기상에서 [SampleChap11]을 오른쪽찰칵하고 [Properties]을 선택한다. [SampleChap11 Property]대화칸이 현시되는데 여기서 [Packaging Configurations]를 선택한다(그림 11-23). 화면오른쪽에서 오른쪽찰칵하고 [Add Archive]를 선택한다.

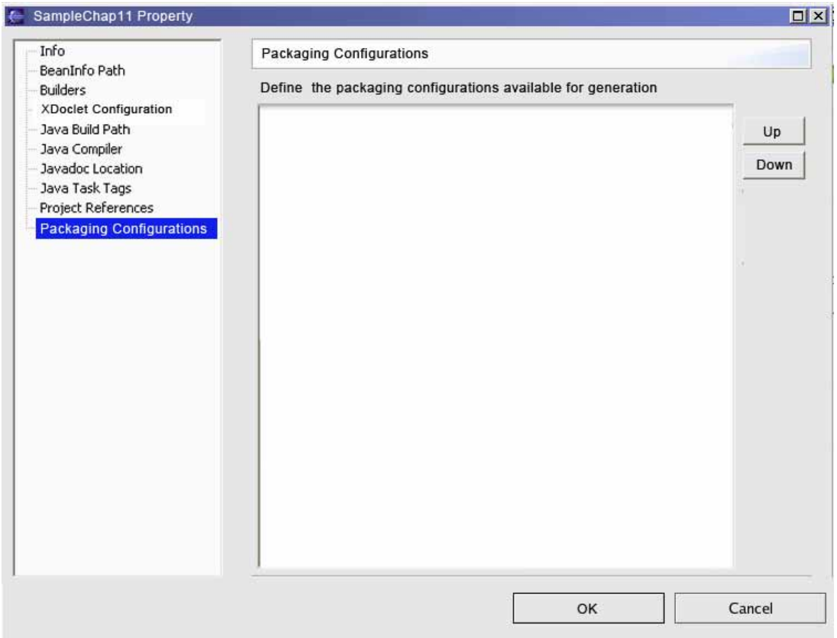


그림 11-23. 패키지화정의대화칸

[Archive Edition]대화칸이 현시되면 [Name]에 《SimpleEJB.jar》를 입력하고 [OK]단추를 찰칵한다(그림 11-24).



그림 11-24. 보존파일명지정대화칸

작성된 [SampleEJB.jar]를 생성하기 위한 정의에 파일 또는 서류철을 추가하여야 한다.(지금까지 작성해놓은 클라스파일은 [SampleChap11/bin]서류철에 들어있다.) 클라스파일을 다음과 같이 추가한다. 화면오른쪽의 [SimpleEJB.jar]상에서 오른쪽찰칵하고 [Add Folder]를 선택한다. [Folder Selection]대화칸이 표시되기때문에 [Project Folder]를 누른다. 그러면 [Select a Folder]대화칸이 표시된다. 《/SampleChap11/bin》을 선택하고 [OK]단추를 찰칵한다(그림 11-25).

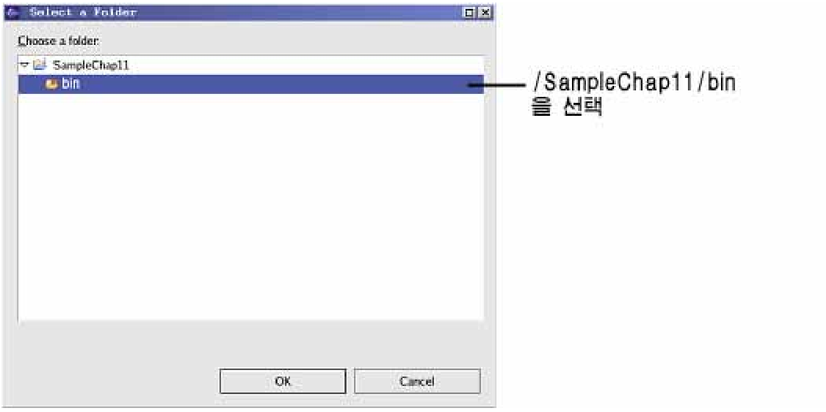


그림 11-25. 서류철지정대화칸

[SampleChap11/bin]서류철을 추가했는데 이것을 그대로 리용하면 [SampleChap11/bin]아래에 있는 클라스가 다 포함되게 되기때문에 EJB로서는 필요없는 클라스파일을 제거해야 한다. 그래서 [includes]에 《sample/ejb/*.class,sample/interfaces/*.class,sample/common/*.class》라고 입력하고 [OK]단추를 찰칵한다. 이렇게 하여 sample.ejb패키지와 sample.interfaces패키지, sample.common패키지안의 클라스만을 포함하도록 한다. 지정할것이 여러개일 때는 반점으로 구분한다(그림 11-26).

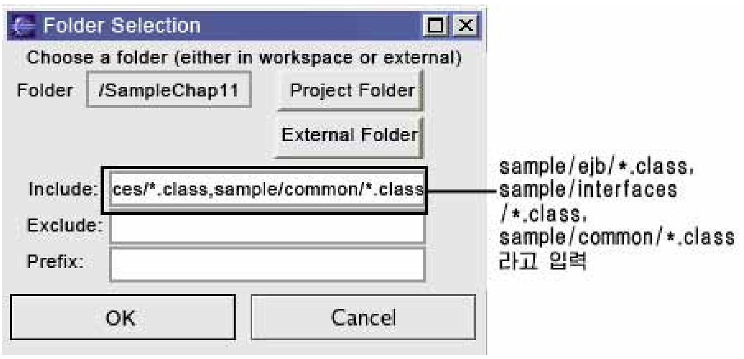


그림 11-26. 서류철추가대화칸

다음으로 EJB배치서술기(ejb-jar.xml, jboss.xml)를 추가한다. [SampleChap11 Properties]에서 [SampleEJB.jar]를 오른쪽찰칵하고 [Add File]를 선택한다. [File Selection]대화칸이 현시되면 [Project File]를 누른다. [Select a File]대화칸이 현시되면 《SampleChap11/src/META-INF/ejb-jar.xml》를 입력 또는 선택하고 [OK]단추를 찰칵한다(그림 11-27).

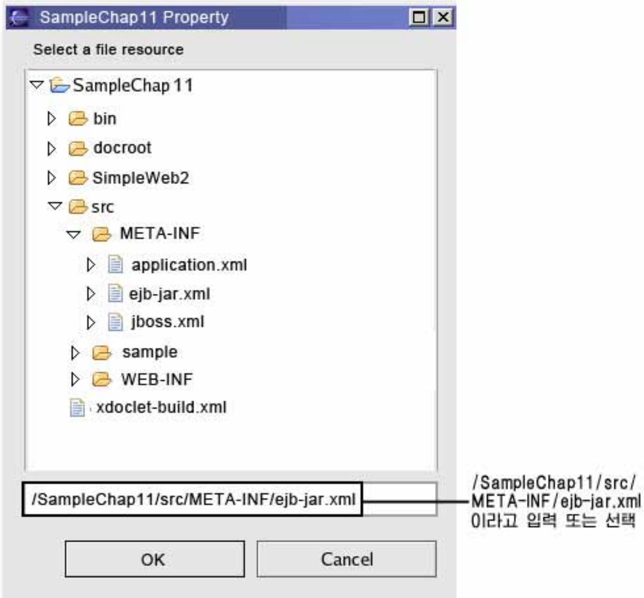


그림 11-27. 파일선택대화칸

[ejb-jar.xml]를 [SimpleEJB.jar]안의 어디에 배치하겠는가를 지정한다. [ejb-jar.xml]는 EJB패키지의 [META-INF]에 배치되기때문에 [Prefix]에 《META-INF》를 입력하고 [OK]단추를 찰칵한다(그림11-28).

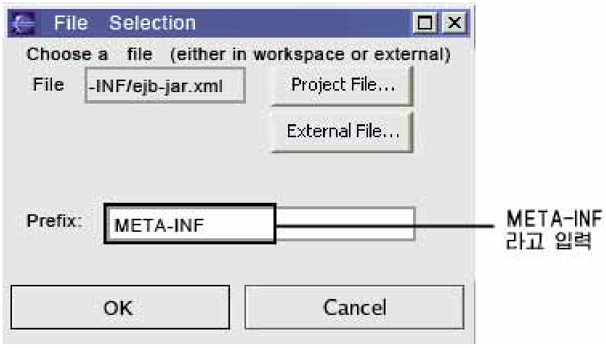


그림 11-28. 파일추가대화칸

다음으로 [jboss.xml]를 추가한다. [ejb-jar.xml]과 같은 절차로 추가를 진행한다. [SampleChap11 Property]에서 [SimpleEJB.jar]를 오른쪽찰칵하고 [Add File]를 선택한다. 다음 [Project File]를 누르고 [SampleChap11/src/META-INF/jboss.xml]를 입력 또는 선택한 다음 [OK]를 찰칵한다. [Prefix]에 《META-INF》를 입력하고 [OK]단추를 찰칵한다. 이것으로 [SimpleEJB.jar]를 생성하기 위한 정의는 끝난다.

EJB의 즉시배치

EJB를 개발하는 과정중에 다른 응용봉사기와 다른 점이 있다. 그것은 대용체파일을 일부러 생성하지 않아도 된다는것이다. WebLogic에서는 ejbc지령, WebSphere에서는 AAT(응용프로그람아쎔블리도구)에 의하여 대용체파일의 생성을 개발자가 책임지고 진행하지만 JBoss에서는 JBoss응용봉사기가 배치시에 자동적으로 해준다. 이것을 즉시배치(hot deployment)라고 하며 개발의 생산성을 향상시킨다.

즉시배치는 JBoss봉사기의 deploy등록부에 jar(또는 war, ear)파일을 복사만하고 봉사기의 재시동이 필요없다. 정상적인 개발과정에 오유수정문을 한행 삽입하고 싶거나 약간의 수정을 가하고 싶다고 할때가 자주 제기된다. 그때마다 봉사기의 재시동이나 배치도구 등을 사용하여 배치를 진행한다면 생산성을 저하시키게 된다. 즉시배치는 생산성향상에 쓸모가 있는 기능이다.

11.4.3. EJB의뢰기용파일의 패키지화

여기서는 EJB의뢰기용으로 배포하기 위한 보존파일의 패키지화를 진행한다. 이 보존파일에는 EJB기준대면부, EJB원격대면부, 검사자료보유클라스가 포함된다. 이 장의 실례응용프로그람에서 EJB의뢰기는 Servlet (MemberSearchServlet2.java)를 나타난다.

정의방법은 EJB요소의 패키지화와 동일하다. [SampleChap11 Property]에서 [Packaging Configurations]를 선택하고 화면오른쪽에서 오른쪽찰칵하여 [Add Archive]를 선택한다. [Archive Edition]대화칸이 나타나면 [Name]에 《SimpleEJBClient.jar》를 입력하고 [OK]단추를 찰칵한다.

다음으로 클라스파일을 추가한다. [SimpleEJBClient.jar]를 오른쪽찰칵하고 [Add Folder]를 선택한다. [Folder Selection]대화칸이 나타나는데 여기서 [Project Folder]를 누르고 《SampleChap11/bin》을 입력 또는 선택한 다음 [OK]단추를 찰칵한다. [Includes]에 《sample/interfaces/*.class, sample/common/*.class》를 입력하고 [OK]단추를 찰칵한다.

이것으로 [SampleEJBClient.jar]를 생성하기 위한 정의가 끝난다.

11.4.4. Web요소의 패키지화

Web요소의 패키지화를 진행한다. 정의방법은 EJB요소의 패키지화와 동일하다. 패키지열람기에서 [SampleChap11]를 오른쪽찰칵하고 [Properties]을 선택한 다음 [SampleChap11 Property]에서 [Packaging Configurations]를 선택한다. 화면오른쪽에서 오른쪽찰칵하고 [Add Archive]를 선택하면 [Archive Edition]대화칸이 나타나는데 [Name]에 《SimpleWeb2.war》를 입력하고 [OK]단추를 찰칵한다.

다음으로 클라스파일을 추가한다. [SimpleWeb2.war]를 오른쪽찰칵하고 [Add Folder]를 선택한다. [Folder Selection]대화칸이 현시되면 [Project Folder]를 누르고《SampleChap11/bin》을 입력 또는 선택하고 [OK]단추를 찰칵한다. [Includes]에《sample/web/*.class》를 입력하고 [Prefix]에 《WEB-INF/classes》를 입력한 다음 [OK]단추를 찰칵한다.

다음으로 Web배치서술기를 추가한다. 우와 마찬가지로 [SimpleWeb2.war]를 오른쪽찰칵하고 [Add Folder]를 선택한다. [Folder Selection]대화칸이 현시되면 [Project Folder]를 누르고《SampleChap11/src/WEB-INF》를 입력 또는 선택한 다음 [OK]단추를 찰칵한다.

JSP, HTML를 추가한다. [Simpleweb2.war]를 오른쪽찰칵하고 [Add Folder]를 선택한다. [Folder Selection]대화칸이 현시되면 [Project Folder]를 누르고 《SampleChap11/docroot》를 입력 또는 선택한 다음 [OK]단추를 찰칵한다. [Folder Selection]에서는 기정설정으로 하고 [OK]단추를 찰칵한다.

마지막으로 EJB의뢰기용JAR를 추가한다. 현단계에서는 [SimpleEJBClient.jar]는 아직 작성되여있지 않지만 정의로서 지정할수는 있다. [SimpleWeb2.war]를 오른쪽찰칵하고 [Add File]를 선택한다. [Project File]를 누르면 [Select a File]대화칸이 나타난다. [Select a file resource]에 [SimpleEJBClient.jar]는 표시되여있지 않지만 화면아래의 입력마당에 《/SampleChap11/SimpleEJBClient.jar》라고 입력한 다음 [OK]단추를 찰칵한다(그림 11-29).

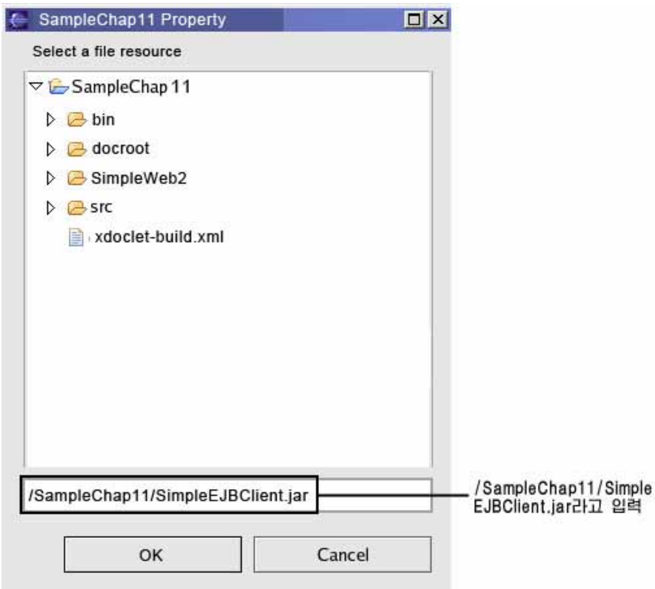


그림 11-29. 파일지정대화칸

[Prefix]에 《WEB-INF/lib》를 입력하고 [OK]단추를 찰칵한다. 이것으로 [SimpleWeb2.war]를 생성하기 위한 정의가 끝난다.

11.4.5. 기업급응용프로그람의 패키지화

기업급응용프로그람을 패키지화한다. 이것도 지금까지 하던 패키지화방법과 같다. 패키지열람기에서 [SampleChap11]을 오른쪽찰칵하고 [Properties]을 선택한다. [Packaging Configurations]를 선택하고 화면오른쪽에서 오른쪽찰칵하며 [Add Archive]를 선택한다.

[Archive Edition]대화칸이 현시되면 [Name]에 《SimpleApp.ear》를 입력하고 [OK]단추를 찰칵한다. 다음으로 [SimpleApp.ear]를 오른쪽찰칵하고 [Add File]를 선택한다. [File Selection]대화칸이 나타나면 [Project File]를 누르고 《SampleChap11/src/META-INF/application.xml》을 입력 또는 선택한 다음 [OK]단추를 찰칵한다. [Prefix]에 《META-INF》를 입력하고 [OK]를 찰칵한다. 같은 절차로 《/SampleChap11/SimpleEJB.jar》와 《/SampleChap11/SimpleWeb2.war》를 추가한다. 이 2개의 파일은 아직 작성되여있는 상태가 아니므로 직접 입력을 해야 한다. 2개의 파일에 대하여 [Prefix]에는 아무것도 입력하지 않는다.

이상으로 모든 설정이 끝났다. 결과 [Packaging Configurations]는 그림 11-30과 같이 설정되였다. [OK]단추를 찰칵하고 설정을 보관해놓는다.

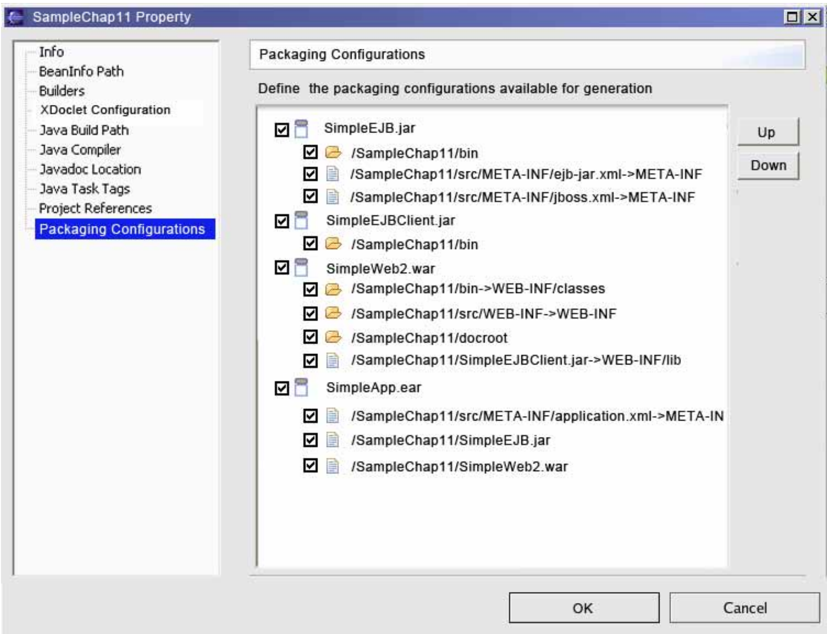


그림 11-30. 입력이 완성된 후 패키지화정의대화칸

11.4.6. 패키지화의 실행

지금까지 진행해온 설정내용으로 패키지화를 진행한다. 패키지열람기상에서 [SampleChap11]을 오른쪽찰칵하고 [Run Packaging]를 선택하면 패키지화가 시작된다. 조종탁보기에는 그림 11-31과 같이 내용들이 나타난다.



그림 11-31. 패키지화실행결과

패키지열람기로 파일이 생성되였는가를 확인해본다. 그림 11-32를 보면 [SimpleApp.ear], [SimpleEJB.jar], [SimpleEJBClient.jar], [SimpleWeb2.war]의 4개파일이 생성된것을 볼수 있다.

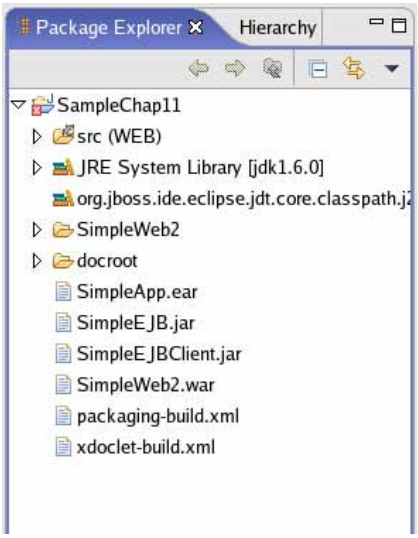
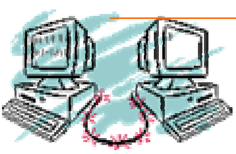


그림 11-32. 생성된 보존파일



제5절. 배치와 실행

작성한 종업원검색응용프로그람을 실행해본다. 실행하기 위해서는 응용프로그람을 JBoss봉사기에 배치하고 JBoss봉사기를 기동해야 한다. 여기서는 JBoss-IDE의 기능을 써서 종업원검색응용프로그람의 배치방법에 대하여 설명한다.

11.5.1. 기업급응용프로그람의 배치

배치방법은 다음과 같다. 패키지열람기에서 배치하려는 파일(여기서는 [SimpleApp.ear])을 오른쪽찰칵하고 [Deployment]→[Deploy To]를 선택한다. [*.ear]파일외에 [*.jar]파일, [*.war]파일에 의한 배치도 가능하다. [Target Choise]대화칸이 현시되면 [JBoss3.2.5(all)[file:/KUT/jboss-3.2.5/server/all/deploy/]]를 선택하고 [OK]단추를 찰칵한다(그림 11-33). 여기서는 《-c all》추가선택으로 JBoss를 기동시킴으로 [JBoss3.2.5(all)]를 선택한다.

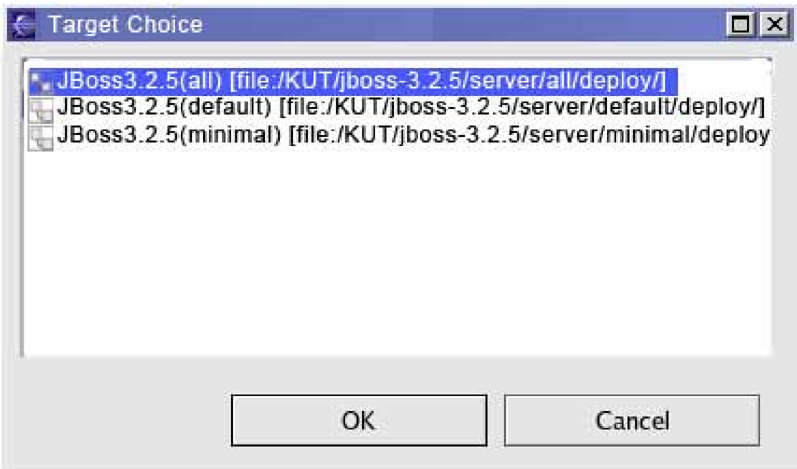


그림 11-33. JBoss봉사기설정대화칸

이상으로 배치가 끝난다. 배치가 끝나면 배치된 파일의 아이콘이 변한다(그림 11-34).

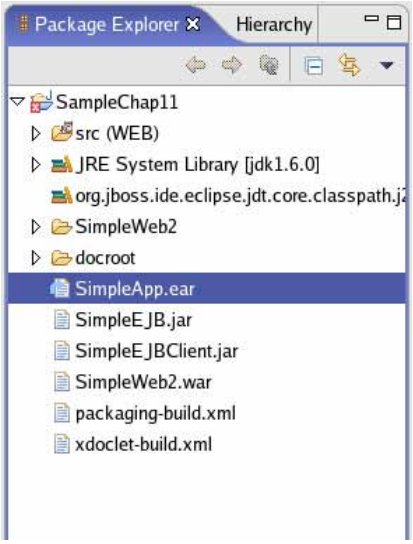


그림 11-34. 배치된 파일

배치해제를 진행하려는 경우에는 패키지열람기에서 [SimpleApp.ear]를 오른쪽찰칵하고 [Deployment]→[Undeploy]를 선택하면 된다. 또한 재배치를 할 때는 [Deployment]→[Redeploy]를 선택한다. 배치, 배치해제를 봉사기의 기동상태에서 진행하면 조종탁보기에 통보가 출력되기때문에 확인할수 있다.

11.5.2. 기업급응용프로그람의 실행

우선 JBoss봉사기를 기동한다. 여기서 다시한번 JBoss봉사기의 기동추가선택으로 《-c all》이 지정되여있는가를 확인하다. 이것이 지정되여있지 않으면 EJB을 실행할수 없다.

JBoss봉사기가 기동되면 열람기를 기동시켜 《http://localhost:8080/SimpleWeb2》라고 입력한다. 성공하면 그림11-35가 현시된다.

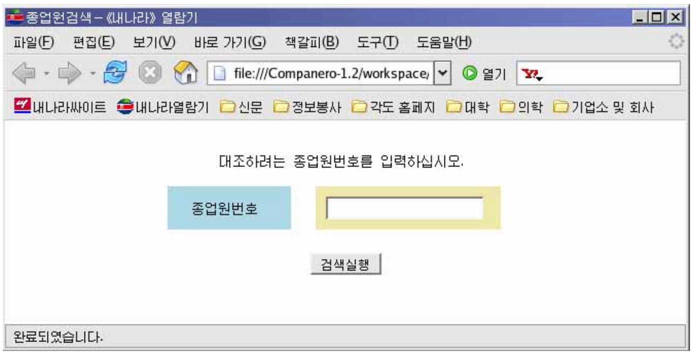


그림 11-35. 종업원검색화면

종업원검색화면에서 종업원번호에《111》라고 입력하고 [검색실행]단추를 찰각한다. 검색처리가 진행되고 그림 11-36과 같이 종업원검색결과화면이 현시된다.

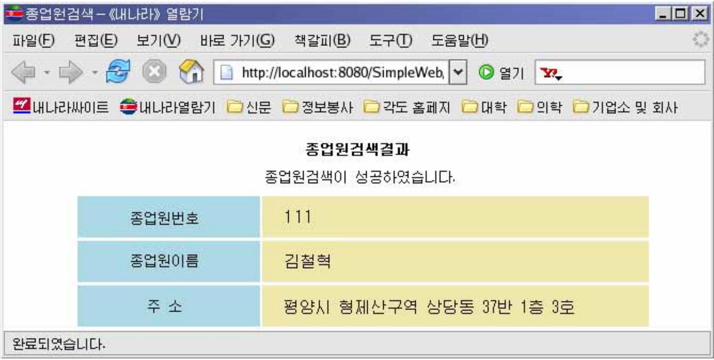


그림 11-36. 종업원검색결과화면

열람기의 되돌이단추를 눌러 다시 종업원검색화면을 현시하고 이번에는《333》을 입력하고 [검색실행]단추를 찰각한다. 그러면 그림 11-37과 같은 종업원검색오유화면이 현시된다. 이상으로 종업원검색응용프로그람의 실행을 확인할수 있다.

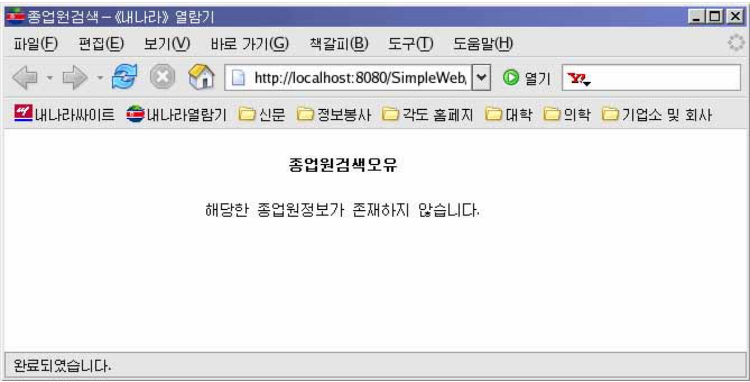


그림 11-37. 종업원검색오유화면

11.5.3. 오유수정

오유수정방법에 대하여 설명한다. 여기서 JBoss봉사기를 기동하고있을 때에는 정지시켜야 한다. 응용프로그람의 실행시에 원천코드상에서 응용프로그람을 일시 정지하고 그때의 상태를 한 걸음씩 확인할수 있다. 그렇게 하려면 JBoss봉사기상에 배치된 2진파일(*.ear파일, *.war파일 등)과 원천코드를 대응시켜야 한다. 왜냐하면 2진파일에는 원천이 포함되지 않았기때문이다.

원천파일의 위치를 Eclipse에서 지정한다. Eclipse의 차림표띠에서 [Run]→[Debug]을 선택한다. 다음으로 [Configurations]에서 [JBoss3.2.5]를 선택하고 화면 오른쪽에서 [Source]태브를 찰칵한다(그림 11-38). 그리고 [Add]단추를 찰칵한다.

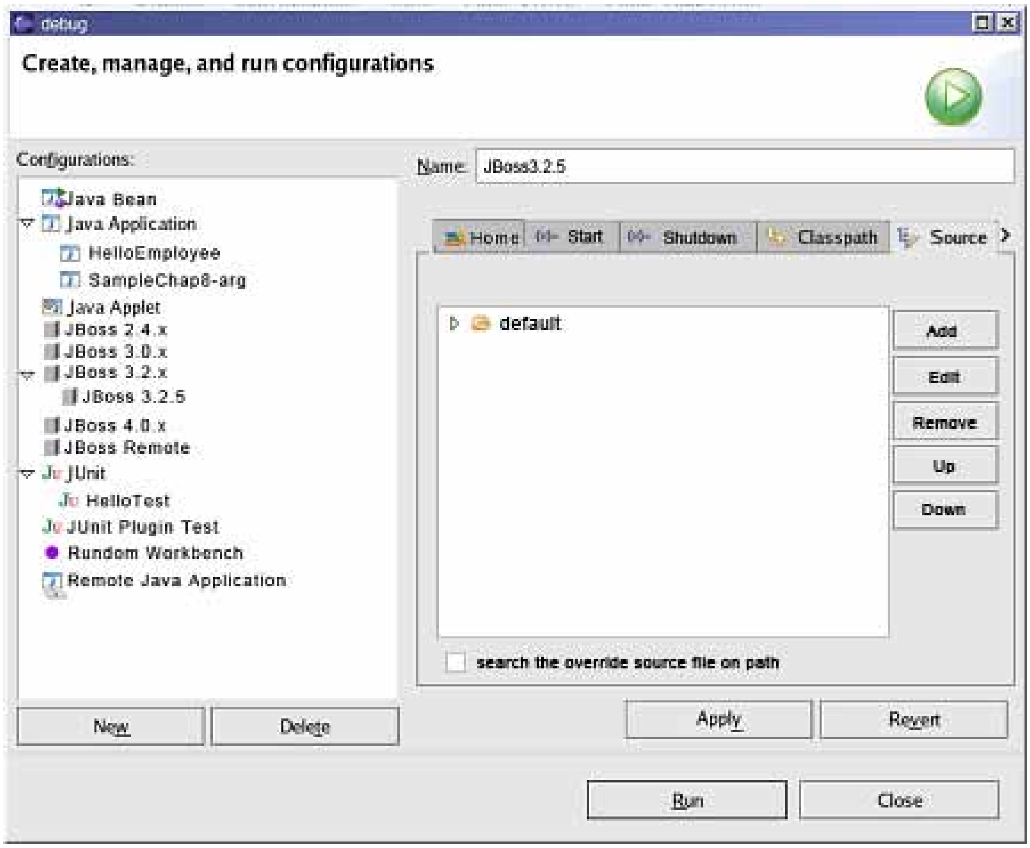


그림 11-38. 원천서류철지정

[Add source]대화칸이 현시되면 [Java Projects]를 선택하고 [OK]단추를 찰칵한다. [Select Projet]대화칸이 현시된다. 원천파일을 추가하려는 프로젝트(여기서는 [SampleChap11])를 선택하고 [OK]단추를 찰칵한다. 이상으로 원천파일지정이 끝난다. [Apply]를 찰칵하고 설정을 보존한다.

다음으로 멈춤점을 설정한다. 실례로 종업원검색Servlet의 오유수정을 진행하기로 한다. [MemberSearchServlet2.java]를 편집기로 연다. 실행을 일시 정지하려는 행의 왼쪽끝에 마우스지시자를 이동시켜 두번찰칵 또는 오른쪽찰칵하여 [Toggle Line Breakpoint]를 설정한다(그림 11-39).

MemberSearchServlet2.java

멈춤점의 설정 → 78

```
70  protected void doPost(
71      HttpServletRequest request,
72      HttpServletResponse response) throws ServletException, IOException {
73
74
75      String resultPage;
76      try {
77
78          String memberId = request.getParameter("memberId");
79
80
81          MemberSearch remote = this.home.create();
82          MemberDataHolder2 mdh = remote.search(memberId);
83
84
85          if (!mdh.isError()) {
86              resultPage = okPage;
87          } else {
88              resultPage = ngPage;
89          }
90
91
92          request.setAttribute("mdh", mdh);
93          javax.servlet.RequestDispatcher dispach =
94              request.getRequestDispatcher(resultPage);
95          dispach.forward(request, response);
96      }
97      catch (Exception e) {
98          javax.servlet.RequestDispatcher dispach =
99              request.getRequestDispatcher(ngPage);
100         dispach.forward(request, response);
101     }
102 }
```

그림 11-39. 멈춤점의 지정

JBoss봉사기를 기동한다. 차림표띠에서 [Run]→[Debug]을 선택하여 현시되는 대화칸에서 [Debug]을 누르면 오유수정방식으로 기동된다.

봉사기가 기동하면 열람기를 열고 《http://localhost:8080/SimpleWeb2》를 입력한다. 앞에서 실행했을 때와 마찬가지로 [종업원번호]에 《111》를 입력하고 [검색실행]단추를 찰칵한다. Eclipse에서 투시절환대화칸이 현시된다. 이것은 방금 지정한 원천코드상에서 오유수정를 하기 위하여 [Debug]투시를 열겠는가를 문의하는 대화칸이므로 [Yes]를 찰칵한다.

제12장. SWT를 리용한 응용프로그람개발

제1절. SWT의 개념

Eclipse는 Swing을 사용한 다른 Java응용프로그람과 대비하면 한결 빠르게 동작한다. 제1장에서도 언급되였지만 그것은 SWT(Standard Widget Toolkit:표준부품도구묶음)라고 부르는 GUI작성용의 도구묶음을 리용하고있기때문이다. SWT는 실행하고있는 조작체계가 가지고있는 본래의 사용자대면에 직접, 그리고 간단히 접근할수 있기때문에 고속으로 그림그리기가 가능하다. Java가상기계에만 고유하여 이식성이 높은 반면에 내부처리가 복잡해지는 Java Swing을 사용한 응용프로그람에 비하면 동작이 실지로 빠르다.

SWT를 리용하기 위한 API는 환경에 관계없이 공통이므로 각 조작체계의 기정사용자대면에 다 접근할수 있다. 조작체계자체의 차이는 Java가상기계에 의해 무시되고 Windows의 경우는 DLL파일, Linux에서는 so파일 등의 기정서고를 경유하여 실행되기때문에 기본적으로 프로그람을 개발할 때 조작체계의 차이를 의식할 필요가 없다. 그래서 원천코드나 클라스파일의 준위에서는 가동기반에 의존하지 않는다고 할수 있다. 그리고 SWT를 사용하면 Windows환경에서는 ActiveX의 리용 등 각 조작체계의 고유한 기술의 리용도 가능하다.

허가는 CPL(Common Public License:일반공개허가)이 적용되므로 가능하다. SWT를 리용하여 응용프로그람을 작성하였다고 해도 원천코드를 공개할 의무가 없으며 개발한 응용프로그람에 대해서 자체의 사용허가권을 적용할수 있다.

제2절. SWT응용프로그람작성

12.2.1. 견본응용프로그람의 개요

간단한 응용프로그람을 만들어 보면서 SWT의 사용법을 보기로 한다.

이 장에서 작성하는것은 달력을 표현하는 응용프로그람이다. 기동하면 그림 12-1과 같은 화면이 현시된다. 단추를 눌러 전달, 다음달의 표시가 가능하고 월을 바꾸면 현시년월을 나타내는 본문칸도 같이 변경된다.

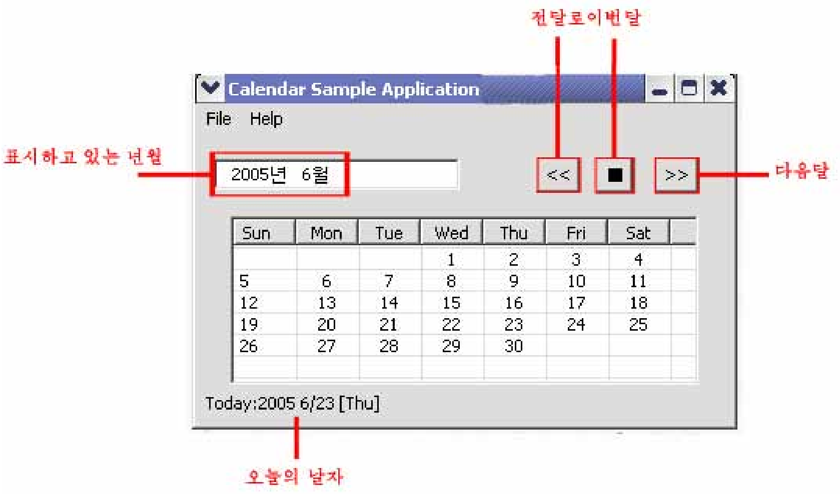


그림 12-1. 달력응용프로그람의 대면부

이 장의 마지막에 SWT와 Swing의 통합에 대해 간단히 취급한다. 우에서 본 SWT응용프로그람을 작성하고 거기에 Swing의 JTree부품을 추가한다(그림 12-2). JTree에서 년과 월을 표시하고 이것을 두번 찰칵하면 지정한 달의 달력으로 변한다.

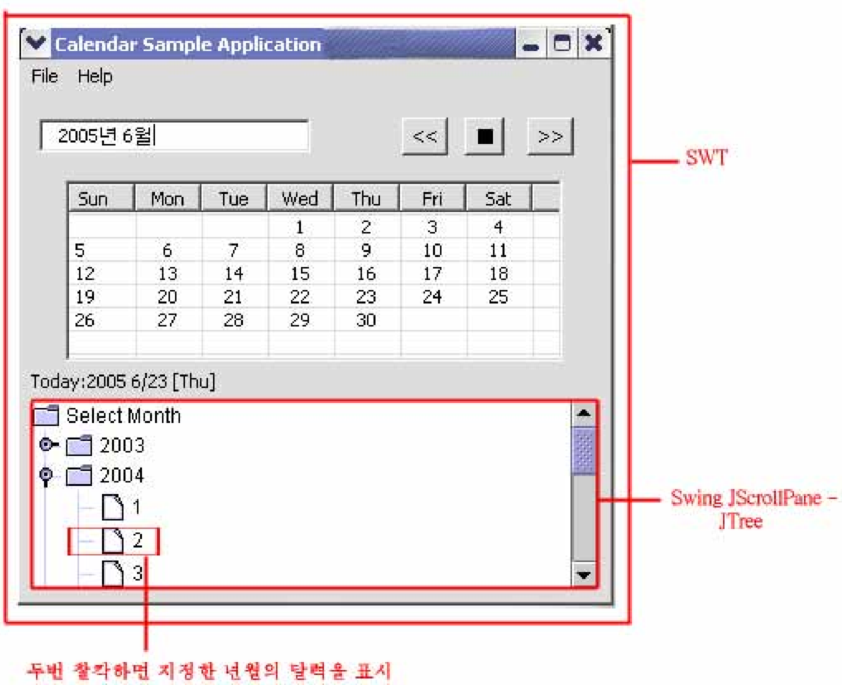


그림 12-2. Swing부품의 통합

12.2.2. 프로젝트의 작성과 개발환경의 설정

먼저 프로젝트를 작성한다.

차림표띠에서 [File]→[New]→[Project]를 선택한다. 새로운 프로젝트를 작성하기 위한 대화칸이 현시되면 [Java Project]를 선택하고 다음으로 넘어간다. 그림 12-3의 대화칸이 현시되는데 [Project name]에는 《SampleChap12》를 입력하고 [Finish]를 찰칵한다.

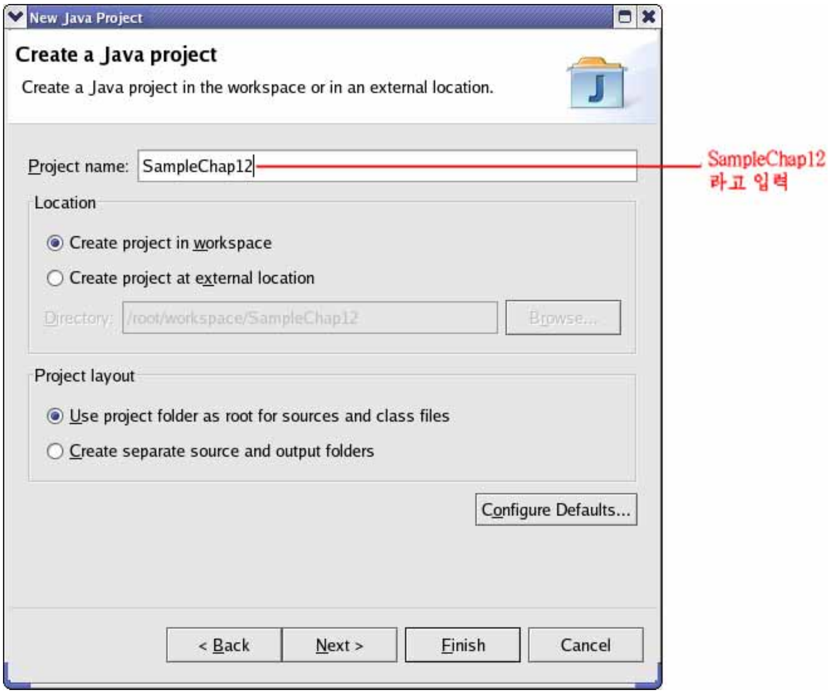


그림 12-3. 프로젝트작성대화칸

다음 SWT의 Java서고 《swt.jar》를 클라스경로에 추가한다. [3.1.4 프로젝트속성]에서 설명한것처럼 클라스경로변수를 설정하여 프로젝트작성경로에 변수이름을 붙여놓는다. 차림표띠의 [Window]→[Preference]를 선택하여 [Preference]대화칸을 열고 [Java]→[Build Path)]→[Classpath Variables]로 전개한다. [New]를 누르면 [New Variable Entry]대화칸이 현시된다. 변수이름에 《SWT_LIB》, 경로는 《/eclipse/plugins/org.eclipse.swt_3.0.1/swt.jar》라고 입력한 다음 [OK]를 찰칵한다(그림 12-4).

동시에 SWT의 원천파일경로에도 변수이름을 붙여놓는다. 이렇게 하면 개발을 하는 과정에 아주 편리하다. 《SWT_LIB》의 경우와 똑같이 [Classpath Variable]화면에서 [New]를 눌러 변수이름에는 《SWT_SRC》를, 경로는《/eclipse/plugins/org.eclipse.platform.source_3.0.1/src/org.eclipse.swt_3.0.1/swtsrc.zip》로 설정한다.

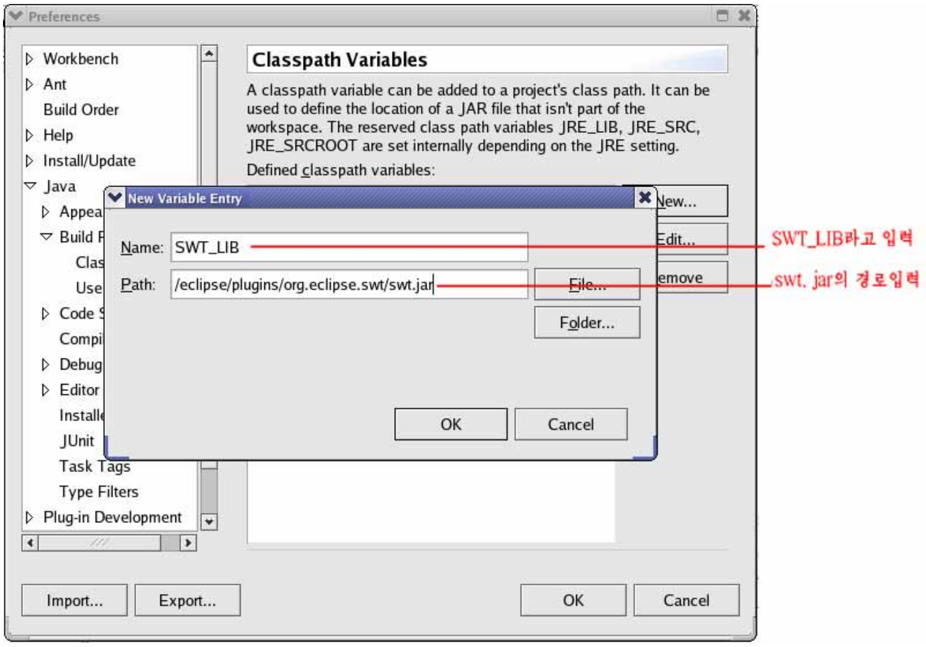


그림 12-4. 클라스경로변수의 새로 작성

계속하여 작성한 이러한 변수를 프로젝트작성경로에 추가한다. [Properties for SampleChap12]대화칸의 [Java Build Path]를 선택하고 [Libraries]태브에서 [Add Variable…]를 찰칵한다. SWT_LIB를 선택하고 [OK]를 찰칵한다(그림 12-5).

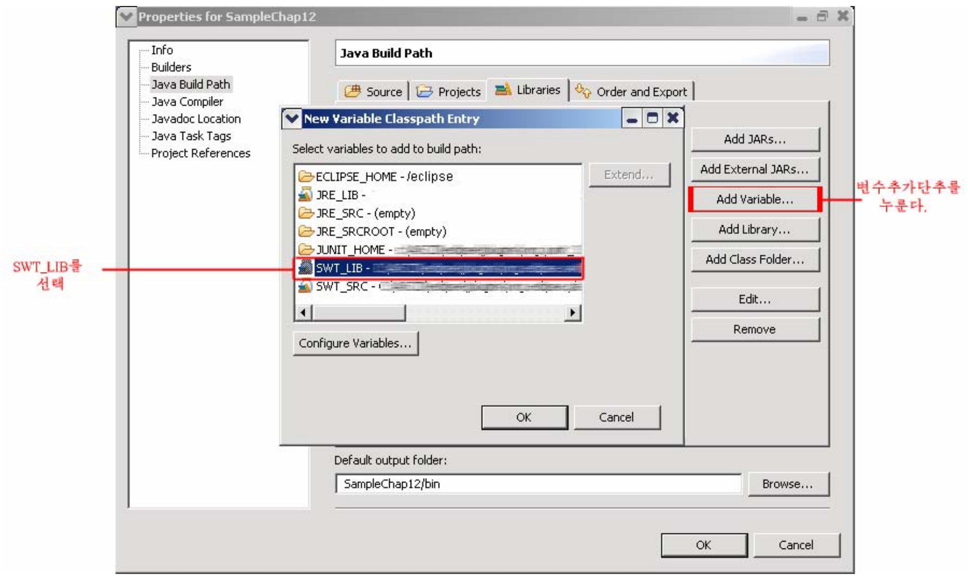


그림 12-5. Java작성경로에 변수 SWT_LIB의 추가

다음 작성된 SWT_LIB를 펼치고 [Source attachment]를 표시한다.이것을 선택한 상태에서 [Edit]를 찰칵하면 [Source Attachment Configuration]대화칸이 나온다. 여기서 [Variable]를 찰칵하고 [Variable Selection]대화칸에서 SWT_SRC를 설정한다. 이상 SWT응용프로그람을 작성할수 있는 환경을 마련하였다.

그리고 SWT응용프로그람의 실행시에는 공유서고(DLL파일)에 경로를 지정해두는것이 필요하다. 이것은 필요한 DLL파일을 프로젝트등록부에 복사하는것으로도 되지만 (/eclipse/plugins/org.eclipse.swt_3.0.1/…/swt-awt

-win32-3063.dll을 프로젝트등록부 eclipse/workspace/SampleChap12에 복사한다.) 이 책에서는 실행시에 VM인수를 설정하는 방법으로 이것을 실현한다. 구체적인 방법은 다음절에서 설명하였다.

12.2.3. 창문의 현시

Shell을 생성하여 가장 간단한 SWT응용프로그람을 작성한다. 먼저 주클라스를 작성한다. [Package Explorer]보기에서 [src]등록부를 선택하여 오른쪽단추를 찰칵하고 [New]→[Class]를 선택하면 [New Java Class]대화칸이 현시된다. [Package]에 《sample.swt》, [Name]에 《CalendarMain》을 입력한다. [Public static void main(String[] args)]의 검사칸을 설정한다(그림 12-6).

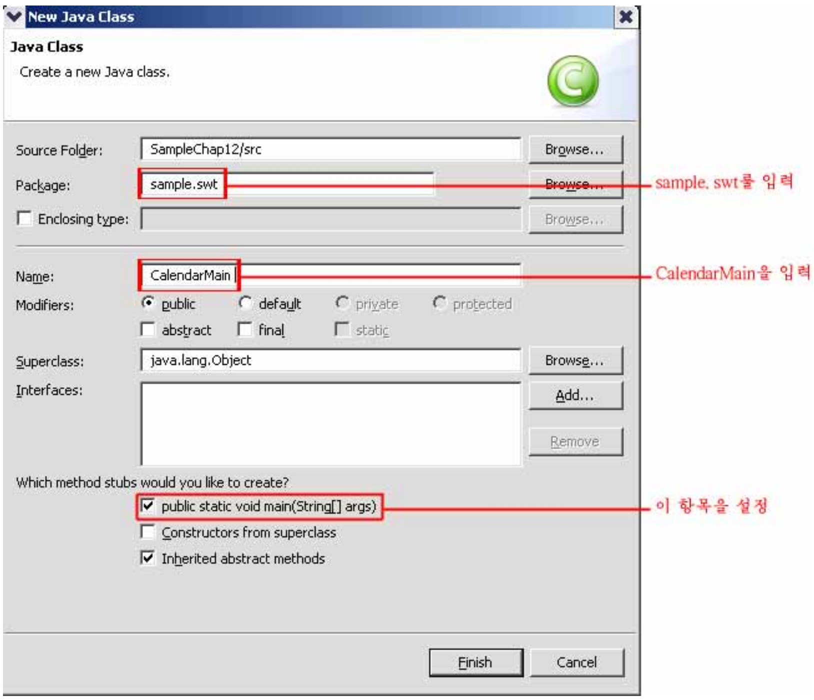


그림 12-6. 주클라스의 작성

창문만을 현시하는 SWT응용프로그람을 작성한다. 우선 Main메쏘드를 목록 12-1과 같이 작성한다.

목록 12-1. 창문을 표시하는 SWT응용프로그람

```
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;

public class CalendarMain {

    public static void main(String[] args) {
            Display display = new Display();        ···(1)
            Shell shell = new Shell(display);       ···(2)

            shell.open();                           ···(3)

            while(!shell.isDisposed()){             ···(4)
                    if(!display.readAndDispatch())
                            display.sleep();
            }
            display.dispose();                      ···(5)
      }
}
```

이제 SWT응용프로그람을 순서대로 따라가면서 구체적으로 보기로 한다. 먼저 전체 스레드를 관리하는 Display구체례를 작성한다(1). 그 다음 Shell의 구체례를 작성하고(2) open메쏘드를 호출한다(3). Shell은 Swing의 JFrame에 해당하는 창문이다. 이 Shell이 사용자의 조작에 의해 닫길 때(isDisposed메쏘드가 참이 될 때)까지 while순환을 반복한다(4). 창문이 닫기면 생성했던 Display구체례를 처분하고 끝낸다(5). 즉 Java가상기계의 자동기억관리(garvage collection)에 맡기는것이 아니라 Display구체례를 명시적으로 처분할 필요가 있는것이다. 그것은 SWT가 순수 Java가 아니기때문이다.

그러면 실행시켜 본다. [Package Explorer]에서 《CalendarMain》클라스를 선택한 상태에서 차림표띠에서 [Run]→[Run]을 선택하면 실행대화칸이 현시된다. 화면왼쪽의 [Configuration]에서 [Java Application]을 선택하고 [New]를 찰칵하면 [Main]태브의 필요한 항목이 자동적으로 입력되여있다(그림 12-7).

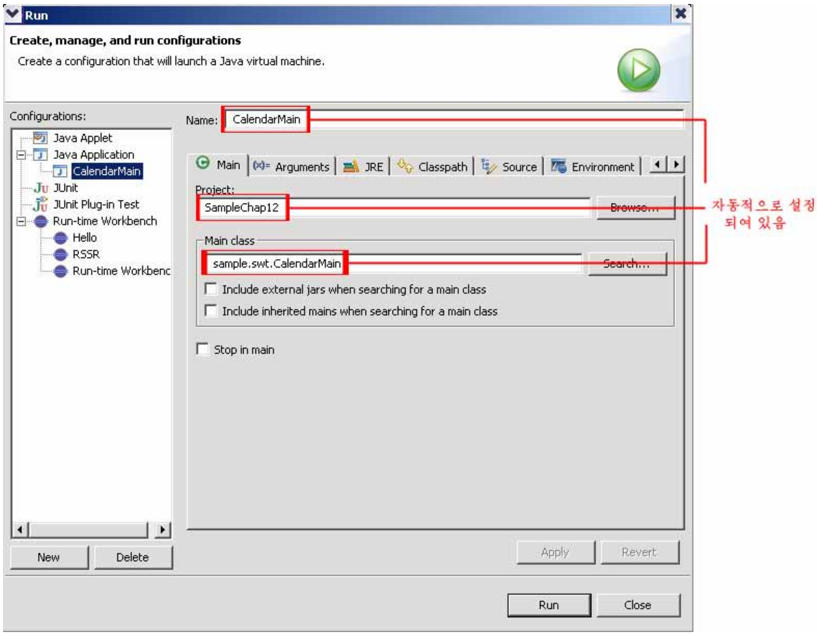


그림 12-7. 기동구성의 설정

계속하여 [Arguments]태브를 선택한다. [VM arguments]에 《-Djava.library.path=/eclipse/plugins/org.eclipse.swt_3.0.1》를 입력한다. 이렇게 하여 지정한 서류철에 있는 공유서고의 경로가 설정된다(그림 12-8).

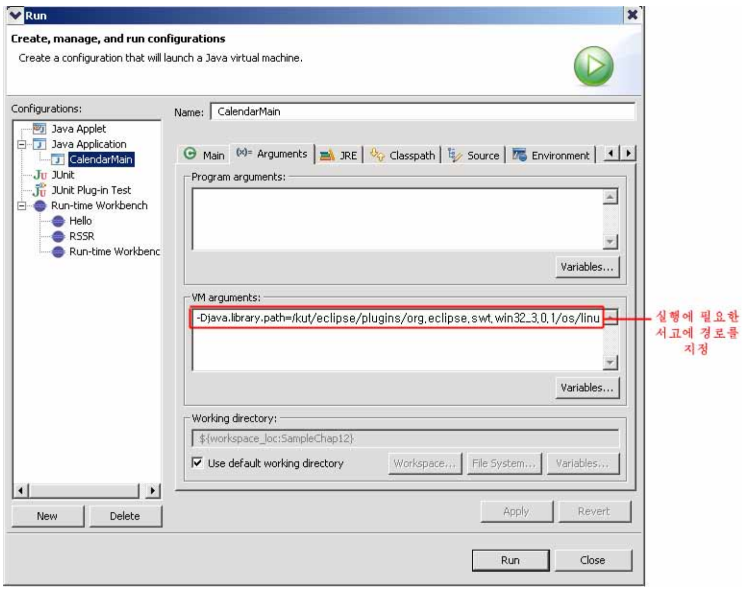


그림 12-8. VM arguments의 설정

[Run]를 찰칵하면 그림 12-9와 같은 창문만 현시하는 SWT응용프로그람이 실행된다.



그림 12-9. 창문을 현시

12.2.4. GUI부품의 작성

작성한 창문에 부분품들을 차례로 추가해나간다. 추가하려는 부분품들을 그림 12-10과 같이 배치한다. **복합체**(Composite)는 여러개의 부품을 내부에 배치하기 위한 부품이다.(Swing에서 말하는 JPanel에 대응된다.)

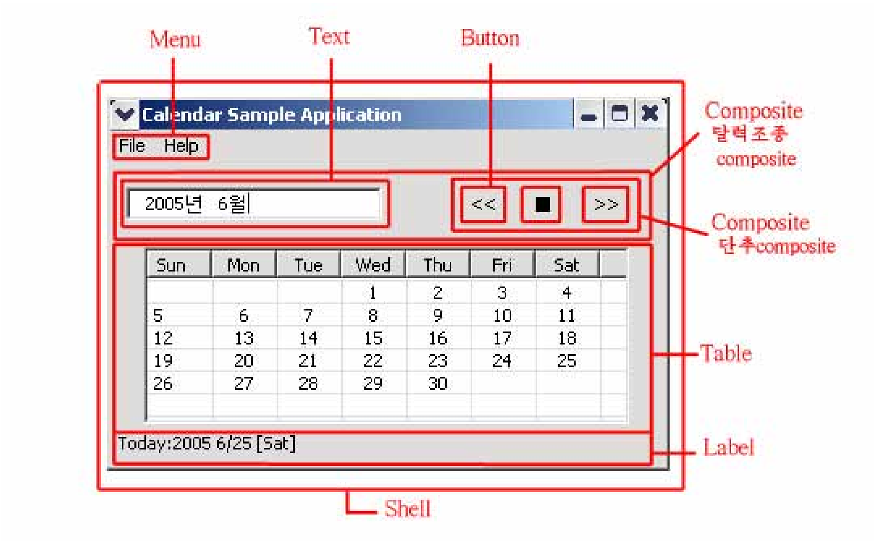


그림 12-10. 추가하려는 부분품

— 놓임새의 설정

GUI응용프로그람을 작성할 때에는 단추의 배치 등 사용의 편리성을 고려하여 화면을 설계하는것이 중요하다. Shell이나 Composite에 부분품을 몇렬로 어떻게 배치하겠는지를 보고 위치나 크기를 지정하는데는 놓임새클라스를 사용한다. 표 12-1에는 표준놓임새에 대하여 보여주었다. 이것들은 추상클라스 Layout의 하위클라스들이다.

표 12-1. 표준놓임새클라스

FillLayout	동일한 크기의 부분품들을 하나의 행 혹은 렬에 배치한다. 가장 간단한 놓임새이며 부분품들사이 공간이나 여백을 지정할수는 없고 간격없이 통채로 사용한다. 여러개의 부분품들이 포함되여있는 경우에는 모든 부분품들이 제일 큰 부분품과 같은 크기로 된다.
RowLayout	FillLayout와 같이 부분품들을 행 혹은 렬에 배치하지만 마당의 설정을 통하여 한 행에 들어가지 못할 때에는 다음 행에 보내기(wrap=true)도 하고 부분품의 크기에 맞추어 자연스러운 크기로 배치(pack=true)하기도 하며 복합체가 커지면 빈 공간을 부분품들사이에 균등하게 분할하여 넓히기(justy=true)도 하는 등의 지정이 가능하다. RowData객체에 의해 각 부분품들의 높이와 폭을 지정할수 있다.
GridLayout	부분품을 격자우에 배치한다. 복잡한 설정시 가장 일반적으로 사용하는 놓임새이다. 몇렬로 부분품을 설치하겠는가를 지정할수 있으며 HTML의 Table과 같이 GridData와 조합하여 가로세로 배치를 정의할수 있다.
FormLayout	FormAttachment를 리용하여 부모합성이나 다른 부분품을 지정한 부분에 접속하는것으로 놓임새를 결정한다. FormData객체를 사용하여 각 부분품을 어떻게 배치하겠는가를 지정한다.

여기서는 GridLayout를 사용한다. 그림 12-10을 본다.

창문의 기본부분을 크게 보면 우에서부터 차례로

1. 달력조종하는 Composite
2. 달력을 표시하는 Table
3. 당일의 날자를 표시하는 Label

이 세부분이 한개 렬에 놓인다. 《달력조종Composite》는 왼쪽에 현재 대면부에 펼쳐진 달력의 년, 월을 가리키는 Text부분이 있고 오른쪽에 달력을 전달 혹은 다음달로 전환시킬수 있는 단추들이 놓이는 《단추Composite》가 배치되였기때문에 2렬의 부분품이 놓인것으로 된다. 《단추Composite》에는 또 3개의 단추가 병렬로 놓였으므로 3렬의 부분품이 놓인것으로 볼수 있다.

먼저 Composite와 Text, Button부분품을 사용하여 《달력조종Composite》의 부분만 창문에 추가한다. 원천코드는 목록 12-2와 같다.

목록 12-2. Text와 Button의 배치

```
package sample.swt;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

public class CalendarMain {
    private Shell shell;
    private Text displayText;
        >>> CalendarTable 성원변수 추가부분 (A)
    private Shell getCalendarShell(Display display){
        shell=new Shell(display);

        shell.setText("Calendar Sample Application");
        shell.setSize(350,250);
        shell.setLayout(new GridLayout(1,true));  (1)
        //달력조종Composite의 작성
        Composite ctlComp=new
              Composite(shell, SWT.NO_FOCUS); (2)
        ctlComp.setLayout(new GridLayout(2,true)); (3)
        GridData compData=new GridData();
        compData.horizontalAlignment=GridData.FILL; (4)
        ctlComp.setLayoutData(compData);

        //Text box의 작성
        displayText=new Text(ctlComp,SWT.SINGLE|SWT.BORDER); (5)
        GridData textData=new
           GridData(GridData.FILL_HORIZONTAL); (6)
        displayText.setLayoutData(textData);

        //단추Composite의 작성
        Composite buttonComp=new Composite(ctlComp,SWT.NO_FOCUS);
        buttonComp.setLayout(new GridLayout(3,true));
        GridData buttonData=new GridData(GridData.FILL);(7)
        buttonData.horizontalAlignment=GridData.END; (8)
        buttonComp.setLayoutData(buttonData);
        //단추의 작성
```

```
            Button button_prev=new Button(buttonComp,SWT.PUSH); (9)
            button_prev.setText("<<");
                 >>> 감시자 추가부분 (B)
            Button button_this_month=new Button(buttonComp,SWT.PUSH);
            Button_this_month.setText("■");
                 >>> 감시자 추가부분  (B)
            Button button_next=new Button(buttonComp,SWT.PUSH);
            button_next.setText(">>");
                 >>> 감시자 추가부분  (B)
                 >>> 표부분품 추가부분      (C)
                 >>> 표식부분품 추가부분    (D)
                 >>> 차림표부분품 추가부분  (E)
            return shell;
        }
        public static void main(String[] args) {
            Display display = new Display();
            Shell shell =new CalendarMain().getCalendarShell(display);

            //쉘을 연다.
            Shell.open();

            while(!shell.isDisposed()){
                 if(!display.readAndDispatch())
                      Display.sleep();
            }
            display.dispose();
        }
    }
```

실행하면 그림 12-11과 같은 화면이 현시된다.

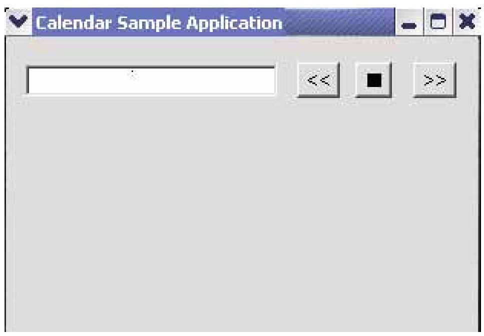


그림 12-11. 달력조종Composite의 추가

이 화면과 목록 12-2를 통해 사용된 부분품에 대하여 자세히 보기로 한다.

쉘에는 1렬로 부분품이 놓이기때문에 GridLayout의 구성자의 첫 인수는 1로 한다(1). 두번째 인수는 true로 지정한다. 이것은 렬의 폭을 자동적으로 등간격으로 하겠는지를 지정하는것인데 한개 렬인 경우는 false로 설정했을 때 렬의 폭이 고정된다. Shell에 Layout을 설정하고 이어서 달력조종Composite를 작성한다(2). 이 Composite의 부모는 쉘이므로 인수에 shell을 설정한다. 달력조종Composite에는 Text와 단추Composite인 두 부분품이 2렬로 놓이기 때문에 GridLayout의 구성자의 첫 인수는 2로 한다(3). 두 부분품의 폭을 같게 하기 위해 두번째 인수는 true로 한다. 달력조종Composite는 쉘이 커지면 거기에 맞추어 수평방향으로 늘어나게 설정한다(4).

GridData객체는 재리용할수 없다는데 주의해야 한다. GridLayout로 관리하고있는 부분품은 모두 유일한 GridData를 가져야 한다. SetLayoutData에서 명시적으로 지정하지 않는 경우에는 기정의 GridData가 생성된다.

— **본문칸**

본문칸을 작성한다(5). 부모Composite는 이미 작성한 달력조종Composite이므로 구성자의 첫인수는 ctlComp이다. 두번째 인수는 격식조종인데 SWT.SINGLE로 1행만 표시하고 SWT.BORDER로 테두리를 준다. Text의 구체례인 displayText는 현시된 달력의 《년, 월》표시를 내부클라스로부터 변경하기 위하여 성원변수로 정의하고있다.

창문크기가 변하면 그에 대응하여 본문칸이 수평방향으로 늘어나도록 설정한다(6). 한편 단추Composite는 크기를 고정하려고 구성자의 인수에 GridData.FILL을 설정하였다(7). 단추Composite를 오른쪽끝에 배치하기 위해 horizintalAlignment(수평배치)마당에 GridData.END를 설정한다(8).

— **단추**

단추부분품의 작성에는 org.eclipse.swt.widgets.Button을 사용한다. 이 클라스 하나에 목록 12-3과 그림 12-12와 같은 절환단추, 검사단추, 라지오단추를 설정한다.

목록 12-3. 단추부분품의 작성

```
Button button1 = new Button(shell,SWT.TOGGLE);
button1.setText("절환단추");

Button button2 = new Button(shell,SWT.CHECK);
button2.setText("검사단추");

Button button3 = new Button(shell,SWT.RADIO);
button3.setText("단일선택단추");
```



그림 12-12. 단추의 목록

목록 12-2에서는 보통단추를 사용하는데 격식조종으로 SWT.PUSH를 지정하였다(9).

— 표

달력부분에서 표를 작성하는데 org.eclipse.swt.widgets.Table을 리용한다. 달력의 표시와 변경 등의 처리를 다른 클라스에서 한다. 목록 12-4에 CalendarTable클라스를 새로 작성하였다.

목록 12-4. CalendarTable클라스

```
import java.util.Calendar;
import java.util.GregorianCalendar;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Table;
import org.eclipse.swt.widgets.TableColumn;
import org.eclipse.swt.widgets.TableItem;

public class CalendarTable {
  Table table;

  //요일의 정의
  static final String[] WEEK =
    { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

    // 현재 표에 표시하고있는 달력
    private Calendar displayCalendar;

    // 오늘의 년, 월, 일, 요일
    int iThisYear;
    int iThisMonth;
    int iThisDate;
```

```
int iThisWeek;
/**
 *
 * @param parent Table구체레의 부모로 되는 부품
 * @param style 격식조종
 */
public CalendarTable(Composite parent, int style) {
   // 현재시각(기정지역, 시간대)으로
   // GregorianCalendar를 구축
   this.displayCalendar = new GregorianCalendar();

   iThisYear = displayCalendar.get(Calendar.YEAR);
   iThisMonth = displayCalendar.get(Calendar.MONTH);
   iThisDate = displayCalendar.get(Calendar.DATE);
   iThisWeek = displayCalendar.get(Calendar.DAY_OF_WEEK);

   table = new Table(parent, style);
   setup();
}

private void setup(){
  // Set Grid Data
  GridData tableGridData = new GridData(GridData.FILL);
  tableGridData.horizontalAlignment = GridData.CENTER;(1)
  tableGridData.verticalAlignment = GridData.CENTER;
  table.setLayoutData(tableGridData);
  // 격자선의 현시
  table.setLinesVisible(true);                    … …(2)
  // 렬의 머리부표시
  table.setHeaderVisible(true);
  // 렬의 머리부에 요일설정
  for (int iCol = 0; iCol < WEEK.length; iCol++) { (3)
     TableColumn col = new TableColumn(table, SWT.CENTER);
     col.setText(WEEK[iCol]);
     col.setWidth(40);
  }
}

private String setCalendar() {
  String strYYMM = "";

 //달력의 내용을 일단 지운다
```

```
    table.removeAll();

    // 현시하려는 년, 월의 문자렬을 작성
    strYYMM = (" " + displayCalendar.get(Calendar.YEAR) + "년" +
        (displayCalendar.get(Calendar.MARCH) + 1) + "월");

    Calendar tmpCal = (Calendar) displayCalendar.clone();
// 날자를 해당한 달의 1일부터 순서대로 표에 설정한다
    tmpCal.set(Calendar.DAY_OF_MONTH, 1);
    int iDispMonth = displayCalendar.get(Calendar.MONTH);

    TableItem item = new TableItem(table, SWT.NULL);

    // 1달분표시
    while (iDispMonth == tmpCal.get(Calendar.MONTH)) {

      // 주가 변하면 TableItem을 추가하여 다음행으로 한다
      if (tmpCal.get(Calendar.DAY_OF_WEEK) ==
          Calendar.SUNDAY) {
            item = new TableItem(table, SWT.NULL);
      }
      item.setText(tmpCal.get(Calendar.DAY_OF_WEEK) - 1,
           Integer.toString(tmpCal.get(Calendar.DATE)));
       // 하루 전진
       tmpCal.add(Calendar.DAY_OF_MONTH, 1);
    }
    return strYYMM;
  }

  public String getTodaysYYMMDD(){
    String strYear = Integer.toString(iThisYear);
    String strMonth = Integer.toString(iThisMonth + 1);
    String strDate = Integer.toString(iThisDate);
    String strDayOfWeek = WEEK[iThisWeek - 1];

    String strYYMMDD =  strYear + " " + strMonth + "/" +
            strDate + " [" + strDayOfWeek + "]" ;
    return strYYMMDD;
  }

  /**
```

```
          * 달력을 다음달로 변경한다. (4)
          *
          * @return 달력에 표시하는 년월의 문자렬
          */
         public String setNextMonth(){
            displayCalendar.add(Calendar.MONTH, 1);
            return setCalendar();
         }

         /**
          * 달력을 전달로 변경한다. (4)
          *
          * @return 달력에 표시하는 년, 월의 문자렬
          */
         public String setPrevMonth(){
            displayCalendar.add(Calendar.MONTH, -1);
            return setCalendar();
         }

         /**
          * 달력을 이번달로 변경 (4)
          *
          * @return 달력에 표시하는 년, 월의 문자렬
          */
         public String setThisMonth() {
            displayCalendar.set(iThisYear, iThisMonth, iThisDate);
            return setCalendar();
         }
      }
```

목록 12-2의 (A)와 (C)자리에 아래의 원천코드와 같은 성원변수의 정의와 CalendarTable의 구체례를 생성하는 부분을 추가한다(두꺼운 글자부분). CalendarTable의 구성자의 인수는 일반적인 SWT부분품과 같이 부모Composite와 격식조종으로 되여있다.

```
      public class CalendarMain {
        private Shell shell;
        private Text displayText;
        private CalendarTable table;
        private Shell getCalendarShell(Display display){
```

```
            ~생략~
            Button button_next=new Button(buttonComp,SWT.PUSH);
            button_next.setText(">>");
            //CalendarTable의 작성
            table=new CalendarTable(shell,SWT.EMBEDDED|SWT.BORDER);
            displayText.setText(table.setThisMonth());
            return shell;
        }
    }
```

CalendarTable클라스에서는 구성자의 인수를 사용하여 Table구체레를 생성한다. Setup메쏘드안에 GridData.setLayoutData를 써서 현시위치를 가운데로 설정하고있다.(목록 12-4(1)) Table에 격자선을 표시하고(2) 렬의 머리부에 요일들을 설정하였다.

이달의 달력을 표시하는 메쏘드와 다음달, 전달의 달력으로 변화시킬수 있는 메쏘드를 각각 만든다(4). 귀환값의 문자렬들은 달력이 현재 표시하고있는 년, 월의 문자렬인데 그것을 Text에 표시한다.

달력의 월, 일표시나 다음달, 전달의 변환 등에는 java.util.Calendar클라스를 리용한다.

— 표식

표식을 작성한다. Label구체레를 생성하고 CalendarTable클라스의 getTodaysYYMMDD메쏘드로부터 얻은 해당한 날의 날자를 setText메쏘드를 리용하여 표식에 설정하도록 한다. 목록 12-2의 (D)위치밑에 다음의 코드를 추가하며 org.eclipse.swt.widgets.Label클라스를 반입한다. Import문을 추가할 때 지름건 Ctrl+Shift+M을 쓰면 편리하다. 표식작성하는 부분을 선택하고 이 지름건을 누르면 [Add Import]대화칸이 나오는데 여기서 org.eclipse.swt.widgets.Label을 선택한다.

```
//Label의 작성
Label label=new Label(shell,SWT.NONE);

//당일의 년월일을 표시
label.setText("Today:"+table.getTodaysYYMMDD());
```

표식은 setFont(Font)로 서체를 설정할수 있고 문자만이 아니라 setImage(Image)에 의해 화상도 삽입할수 있다. Text와 같이 격식조종을 사용하여 테두리의 설정 등도 가능하다.

— 감시자

단추에 사건을 추가하여 CalendarTable클라스의 표시달을 변화시키는 메쏘드를 호출한다. 목록 12-2(B)위치에 아래의 코드에서 굵은 글자로 쓴 부분을 써넣어 단추부분품의 구체례에 Listener(감시자)를 추가한다.

org.eclipse.swt.widgets.Listener클라스와 org.eclipse.swt.widgets.Event클라스를 반입한다. button_prev단추가 눌리는 사건처리에서 setPrevMonth()를 호출하고 귀환값을 displayText에 설정한다. button_this_month의 경우는 CalendarTable의 setThisMonth()를, button_next인 경우 CalendarTable의 setNextMonth를 호출한다.

```
Button button_prev=new Button(buttonComp,SWT.PUSH);
button_prev.setText("<<");
button_prev.addListener(SWT.Selection, new Listener() {
        public void handleEvent(Event e) {
        displayText.setText(table.setPrevMonth());
        }
});
Button button_this_month=
        new Button(buttonComp,SWT.PUSH);
button_this_month.setText("■");
button_this_month.addListener(SWT.Selection, new Listener()
{
        public void handleEvent(Event e) {
        displayText.setText(table.setThisMonth());
        }
});
Button button_next=
         new Button(buttonComp,SWT.PUSH);
button_next.setText(">>");
button_next.addListener(SWT.Selection, new Listener() {
         public void handleEvent(Event e) {
         displayText.setText(table.setNextMonth());
         }
});
```

— 차림표

먼저 Listener구체례를 생성하고 차림표에 추가하려는 사건들을 정의한다. 목록 12-5를 목록 12-2(E)자리에 추가한다. 다음과 같은 import문이 필요하다.

```
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.MenuItem;
import org.eclipse.swt.widgets.MessageBox;
```

목록 12-5. 차림표의 작성

```
Listener exitListener = new Listener() {
  public void handleEvent(Event e) {
     MessageBox dialog = new MessageBox(shell, SWT.OK |
             SWT.CANCEL| SWT.ICON_QUESTION); (1)
     dialog.setText("Question");
     dialog.setMessage("Exit?");
     if (e.type == SWT.Close)
        e.doit = false;
     if (dialog.open() != SWT.OK)
        return;
     shell.dispose();
  }
};
Listener aboutListener = new Listener() {
  public void handleEvent(Event e) {
     MessageBox dialog = new MessageBox(shell, SWT.OK|
        SWT.ICON_INFORMATION); (2)
     dialog.setText("About");
     dialog.setMessage("Eclipse Calendar Sample Application.");
     if (dialog.open() == SWT.OK)
        return;
     shell.dispose();
  }
};
//차림표
Menu mb = new Menu(shell, SWT.BAR); (3)
shell.setMenuBar(mb); (4)
MenuItem fileItem = new MenuItem(mb, SWT.CASCADE);(5)
MenuItem helpItem = new MenuItem(mb, SWT.CASCADE);

FileItem.setText("&File");
Menu fileMenu = new Menu(shell, SWT.DROP_DOWN); (6)
fileItem.setMenu(fileMenu); (7)
MenuItem exitItem = new MenuItem(fileMenu, SWT.PUSH); (8)
exitItem.setText("&Exit\tCtrl+X");
exitItem.addListener(SWT.Selection, exitListener); (9)
exitItem.setAccelerator(SWT.CONTROL + 'X'); (10)

helpItem.setText("&Help");
Menu helpMenu = new Menu(shell, SWT.DROP_DOWN);
helpItem.setMenu(helpMenu);
MenuItem aboutItem = new MenuItem(helpMenu, SWT.PUSH);
aboutItem.setText("&About\tCtrl+A");
```

```
    aboutItem.setAccelerator(SWT.CONTROL + 'A');
    aboutItem.addListener(SWT.Selection, aboutListener);
```

사건감시자의 처리를 진행할 때 코드를 읽거나 객체의 재리용성을 높이는 측면에서 이름이 없는 클라스는 피하는것이 좋다. exitListener는 통보칸을 현시하여 응용프로그람을 끝내기하겠는가를 사용자에게 물어본다. MessageBox구체례의 생성시에 통보칸을 질문형식으로 만드는데 OK(확인), Cancel(취소)를 표시하는 등 속성을 지정한다(1). AboutListener도 마찬가지로 MessageBox구체례를 생성하지만 구성자의 인수에서 격식조종에는 SWT.OK|SWT.ICON_INFORMATION을 설정하여 정보대화칸을 표시한다(2).

감시자를 생성하고 계속하여 차림표부분품을 만든다. 차림표띠를 생성(3)하고 shell에 설정한다(4). 생성한 차림표띠에 차림표항목을 작성한다(5). 다음은 누르면 아래로 펼쳐지는 차림표를 작성하여(6) 이것을 차림표항목에 설정한다(7). 그리고 최종적으로 선택하기 위한 차림표항목을 작성한다. 이것은 (6)에서 작성한 부분차림표를 포함한다(8). 이 차림표에 Listener를 추가한다(9). 매개 차림표항목에는 (10)과 같이 간단히 지름건을 추가할수 있다. 이상으로 그림 12-1에서 보여준 견본응용프로그람을 작성하였다.

제3절. Swing부품과의 통합

Swing의 부품을 결합하여 SWT응용프로그람을 작성할수 있다.

앞에서 만든 달력프로그람에 Swing부품을 추가해본다. Jtree를 리용하여 년, 월을 계층구조로 표시한다. 감시자를 붙여서 두번 찰칵하면 지정한 년, 월의 달력으로 변하게 한다.

먼저 CalendarTable클라스에 년, 월을 직접 지정하여 달력의 현시를 변화시키는 메쏘드 setSpecifiedMonth를 추가한다.

목록 12-6. CalendarTable에 년, 월을 지정하는 메쏘드를 추가

```
/**
* @param iYear  현시하려는 년
* @param iMonth 현시하려는 달
* @return      달력에 표시하는 년, 월의 문자렬
*/
public String setSpecifiedMonth(int iYear, int iMonth){
    displayCalendar.set(iYear, iMonth, iThisDate);
    return setCalendar();
}
```

CalendarMain클라스는 javax.swing.JTree부품을 작성하여 마우스감시자를 추가한다. TreeNode를 두번 찰칵하여 선택한 년, 월을 얻어 setSpecifiedMonth를 호출한다. 다음의 감시자(목록 12-7)를 getCalendarShell메쏘드의 제일 마지막에 있는 《return shell;》문앞에 추가한다. 필요되는 import문도 지름건 Ctrl+Shift+M을 리용하여 추가한다.

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JScrollPane;
import javax.swing.Jtree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;
import org.eclipse.swt.awt.SWT_AWT;
```

getCalendarShell의 제일 앞에 shell의 크기를 setSize메쏘드에 의해 지정하고 있지만 Jtree를 넣을만한 크기만 필요하므로 《shell.setSize(350,400);》으로 변경한다.

목록 12-7. Swing부품의 추가

```
// Swing부품의 부모인 Composite
Composite swingComp = new Composite(shell, SWT.EMBEDDED);

swingComp.setLayout(new GridLayout(1, true));
GridData swingCompData = new GridData(GridData.FILL_BOTH);
swingComp.setLayoutData(swingCompData);

// 새로운 AWT Frame을 작성
java.awt.Frame awtFrame = SWT_AWT.new_Frame(swingComp);

java.awt.Panel panel = new java.awt.Panel(new java.awt.BorderLayout());
awtFrame.add(panel);

DefaultMutableTreeNode root =
    new DefaultMutableTreeNode("Select Month");
DefaultMutableTreeNode nodePreYear =
    new DefaultMutableTreeNode("2003");
DefaultMutableTreeNode nodeThisYear =
    new DefaultMutableTreeNode("2004");
DefaultMutableTreeNode nodeNextYear =
    new DefaultMutableTreeNode("2005");

root.add(nodePreYear);
root.add(nodeThisYear);
root.add(nodeNextYear);
```

```
    for (int iMonth = 1; iMonth < 13; iMonth++) {
      nodePreYear.add(new
        DefaultMutableTreeNode(Integer.toString(iMonth)));
      nodeThisYear.add(new
        DefaultMutableTreeNode(Integer.toString(iMonth)));
      nodeNextYear.add(new
        DefaultMutableTreeNode(Integer.toString(iMonth)));
    }

    final JTree swingJtree = new JTree(root);
    swingJtree.setDoubleBuffered(true);

    // 마우스감시자
    MouseListener ml = new MouseAdapter() {
       public void mousePressed(MouseEvent e) {
          TreePath selPath = swingJtree.getPathForLocation(e.getX(),e.getY());

          if (e.getClickCount() == 2 && selPath != null &&
                        selPath.getPathCount() == 3) {
             Object[] selMonth = new Object[3];
             selMonth = selPath.getPath();

             final int iYear =
             Integer.valueOf(selMonth[1].toString()).intValue();
             final int iMonth =
               Integer.valueOf(selMonth[2].toString()).intValue() - 1;

             shell.getDisplay().syncExec(new Runnable(){    (1)
               public void run() {
               // 달력을 지정한 년월로 변경한다.
               displayText.setText
                   (table.setSpecifiedMonth(iYear, iMonth));
               }
             });
          }
       }
    };
    swingJtree.addMouseListener(ml);
    JScrollPane scrollPane = new JScrollPane(swingJtree);
    panel.add(scrollPane);
```

Swing Listener로부터 SWT의 GUI를 변경한다. 이때 다음과 같은것에 주의해야 한다. SWT응용프로그람은 쉘을 닫을 때까지 사건순환을 실행한다(목록 12-1(4)). 이

동안에 마우스조작 등이 진행된 경우 조작체계로부터 GUI사건을 대기렬로 받고 처리를 진행한다. 이것을 응용프로그람의 《UI스레드》라고 부른다. 모든 사건은 응용프로그람의 UI스레드가 방아쇠(trigger)로 되여있으므로 사건을 처리하는 응용프로그람코드는 부분품에로의 자유로운 접근이 허가되여 특별한 조작없이 GUI의 변경 등이 가능하다. 여기서 원래 상정되여있지 않은 Swing 등의 비UI스레드로부터 SWT의 UI스레드가 직접 호출되면 SWT레외가 발생한다.

이것을 피하기 위하여 display.syncExec메쏘드에 Runnable의 구체레를 넘긴다. Runnable의 run메쏘드를 경유하여 SWT의 부분품을 변경한다(목록 12-7(1)). SyncExec는 호출하는 쪽의 스레드를 차단하여 적절한 시기에 UI스레드안에서 지정한 처리를 실행한다.

Swing부품을 SWT응용프로그람에서 사용하는 경우 실례로 새로 작성한 SWT응용프로그람에 기존의 Swing응용프로그람의 일부를 재리용한다는 점에서 생각해본다. 이 절에서 소개한것처럼 Swing과 SWT는 통합할수 있지만 사건의 처리가 힘들고 복잡하다. 가볍고 고속성을 보장할수 있다는 리유로 응용프로그람을 SWT로 작성하는것이지만 응답향상의 측면으로부터 기존의 Swing부품을 순차적으로 SWT에로 이행하는 편이 낫다. SWT와 Swing이 완전히 1대1로 대응되지 않으므로 Swing과 같이 가동기반을 쓰지 않고 완전히 동일한 모양과 기능을 제공한다는것은 SWT에서 불가능하다. 대표적으로 부품에 대하여 Swing과 SWT의 대응관계는 표 12-2와 같다.

표 12-2. Swing과 SWT의 대응

	Swing	SWT
창문모양	JFrame	Shell
	JPanel	Composite
	JMenu, JMenuBar	Menu
	JToolBar	ToolBar, CoolBar
	JScrollBar	ScrollBar
입력	JButton	Button+SWT.PUSH
	JToggleButton	Button+SWT.TOGGLE
	JCheckBox	Button+SWT.CHECK
	JRadioButton	Button+SWT.RADIO
	JComboBox	Combo
	JFileChooser	FileDialog
	JFileChooser	DirectoryDialog
	JTextField	Text+SWT.SINGLE
	JTextArea	Text+SWT.MULTI
	JSlider	Slider
표시	JLabel	Label
	JList	List
	JTree	Tree
	JTable	Table

제13장. 끼워넣기개발의 초보

이 장에서는 끼워넣기개발의 초보에 대하여 소개한다.

제1절. 끼워넣기의 구조

Eclipse본체에는 확장점(extension point)이라고 하는 대면이 있다. 이 대면을 실현한 클라스를 **확장클라스**라고 한다. 그리고 어느 클라스가 확장되였는가를 Eclipse의 끼워넣기기구에 전달하는 파일이 끼워넣기명시(plugin.xml)이다. Eclipse에서 끼워넣기라는것은 확장점에서 정의된 대면의 실현클라스와 끼워넣기명시를 조합한것이라고 말할수 있다. 엄밀한 정의를 한다면 좀 어렵고 초기에는 이 정도의 리해로 시작한다. 간단한 끼워넣기를 만들어보면서 끼워넣기의 구조를 본다.

제2절. Hello끼워넣기

Eclipse끼워넣기는 Eclipse를 사용하여 개발한다.

Eclipse의 전신인 IBM VisualAge for Java본체는 Smalltalk로 개발되였다. 그래서 Eclipse를 개발하고있는 IBM에 있어서 Java로 IDE를 만든다는것이 혁신적인것이였다. 끼워넣기를 개발할 때 [Plug in Development Perspective](그림 13-1)을 사용하면 편리하다. 이 투시는 Java투시와 비슷하고 사용방법도 크게 다른것이 없다.

이제부터 만드는 끼워넣기는 [Hello끼워넣기]라고 부르는데 Eclipse가 제공하는 간단한 견본이다.

1. 작업대의 도구띠에서 [New]단추를 선택하고 [Plug-in Development]→[Plug-in Project]를 선택한 다음 [Next]를 찰칵하여 [New Plug-in Project]대화칸을 연다.

2. 그림 13-2와 같이 프로젝트이름에 [kp.ac.kut.example.hello]라고 입력한다. 다른 사람이 작성한 끼워넣기와 혼돈하지 않기 위해 프로젝트의 이름달기는 이처럼 령역이름형식으로 하였다. 그리고 다음에 나오는 끼워넣기ID도 프로젝트이름과 같은것을 사용한다. [Next]를 찰칵한다.

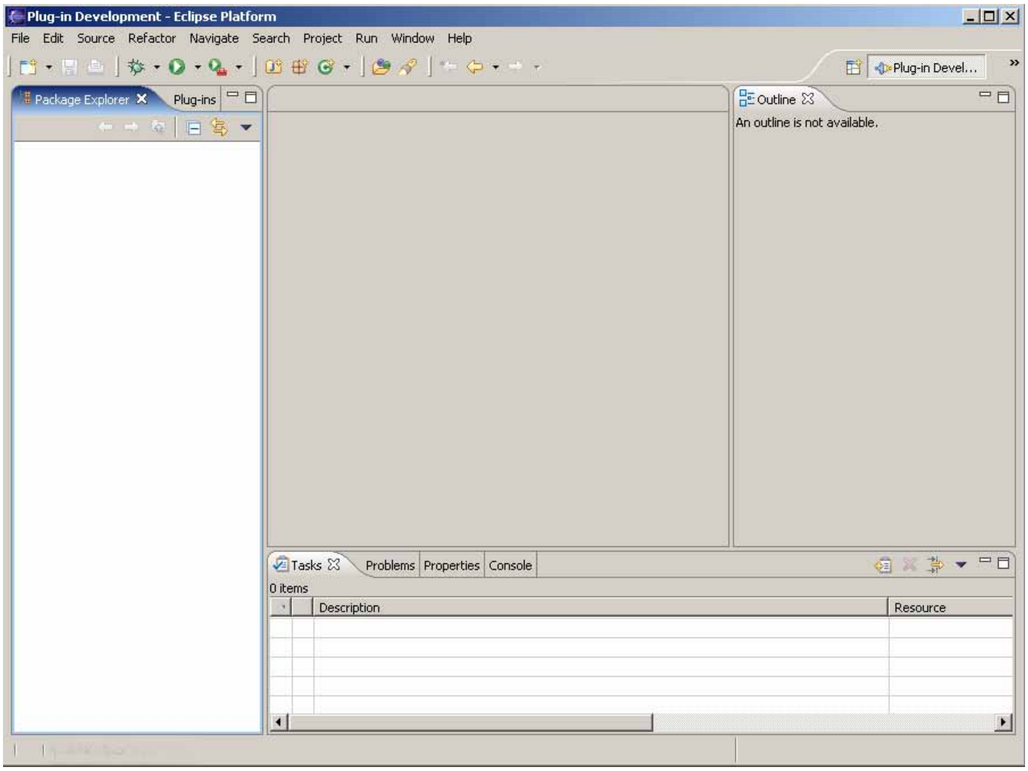


그림 13-1. 끼워넣기개발투시

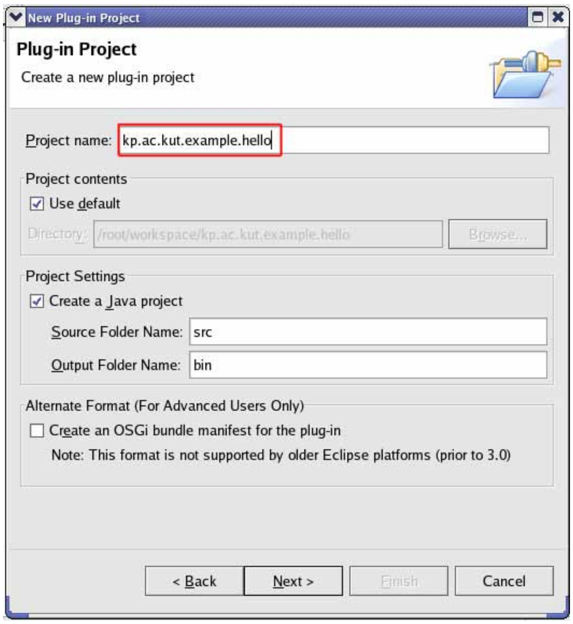


그림 13-2. 새 끼워넣기프로젝트

3. [Plug-in Properties]안의 끼워넣기이름, 끼워넣기판번호, 끼워넣기제공자, 클라스이름을 입력한다. 여기서는 끼워넣기제공자에 《KUT》를 입력하고 판번호를 0.0.1로 한다. 이 외에는 자동적으로 입력되여있는 값을 그대로 사용한다. 즉 끼워넣기ID는 프로젝트이름과 같다. 그리고 [Plug-in Class]안의 두개의 검사칸 [Generate the java class that controls the pulg-in's life cycle]과 [The pulg-in will contributions to the UI]를 선택한다. 마지막에 [Next]를 찰칵한다.(그림 13-3)

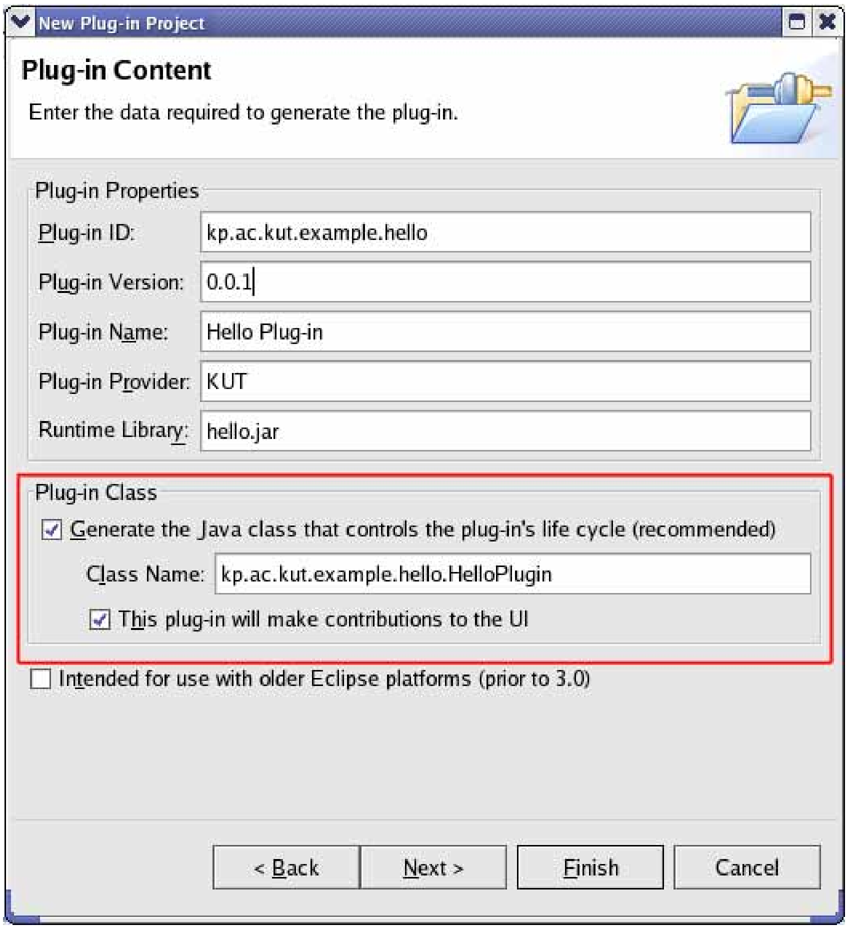


그림 13-3. 끼워넣기프로젝트구조

4. 코드생성조수를 리용하여 코드를 생성한다. 여기서는 [Create a pulg-in using one of the templates]을 선택하여 [Hello World]를 선택한 다음 [Next]를 찰칵한다(그림 13-4).

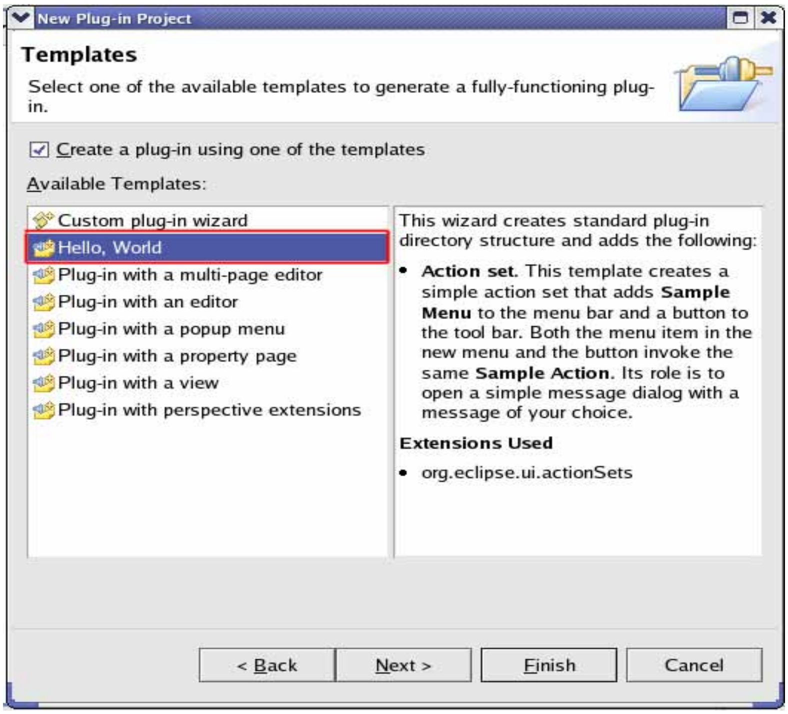


그림 13-4. 끼워넣기코드생성

5. [Sample Action Set]대화칸은 Hello World고유의 정보를 입력한다. Hello World는 차림표띠나 도구띠에 단추를 추가하는 끼워넣기의 형판이다. 그래서 Action클라스이름도 입력하고 단추를 눌렀을 때 현시할 통보문도 입력한다(그림 13-5).

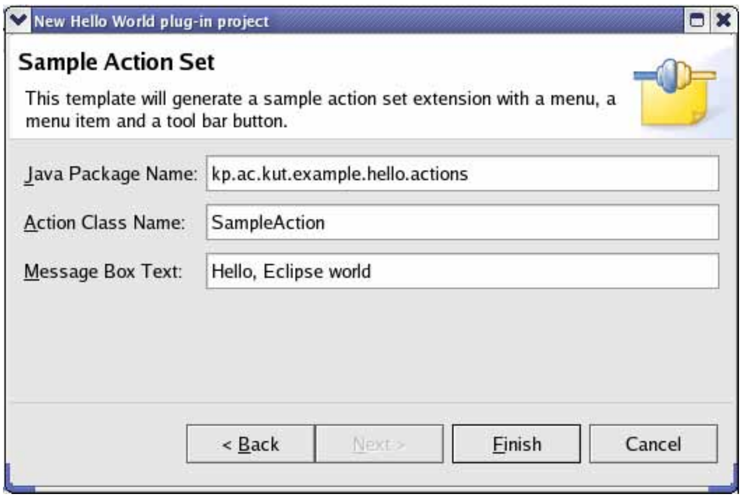


그림 13-5. Sample Action Set대화칸

6. [Finish]를 찰칵한다.

새 끼워넣기가 작성되면 작업대에 [Overview]라는 페지가 나타난다(그림 13-6). 이렇게 Hello끼워넣기가 완성된다. 이제는 끼워넣기를 기동시켜 본다.

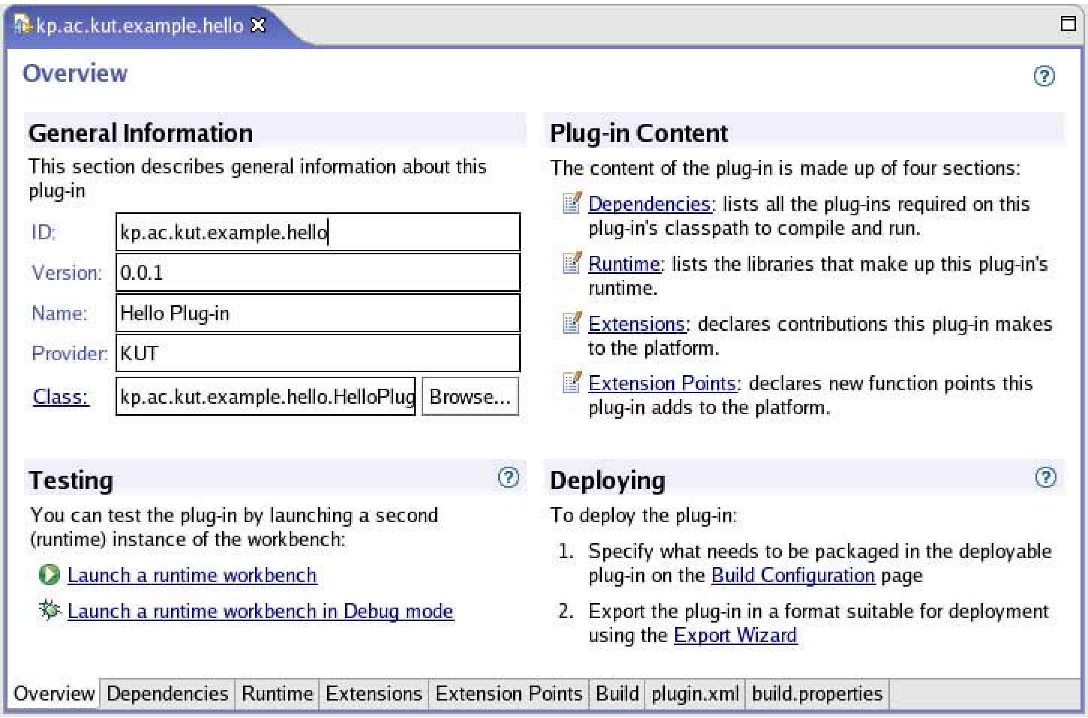


그림 13-6. Overview페지

그러자면

1. [Package Explorer]보기에서 [kp.ac.kut.example.hello]를 선택한다.

2. 차림표띠에서 [Run]→[Run]을 선택하면 실행대화칸이 기동한다. [Run-time Workbench]를 선택하고 [Name]에는 《Hello》를 입력한다. 그다음 왼쪽아래에 있는 [New]를 찰칵한다(그림 13-7). 마지막에 [Run]단추를 눌러 실행한다.

3. 그러면 새로운 작업대가 기동한다. 오른쪽우의 최소화단추를 눌러 [Welcome]창을 작게 한다.

그리고 Hello끼워넣기가 어디에 매몰되는지 살펴본다. 먼저 볼것은 도구띠에 있는 [Hello Eclipse World]단추이다. 그 다음은 차림표에 있는 [Sample Menu]이다(그림 13-8). 어느것으로 실행해보든 결과는 둘다 그림 13-9와 같다.

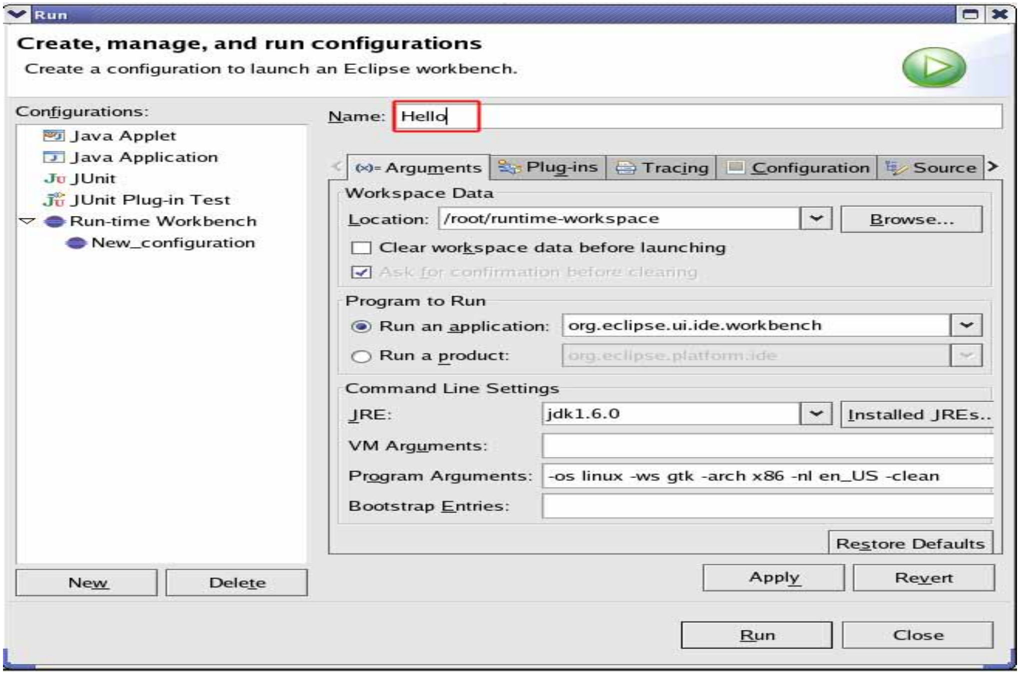


그림 13-7. 기동대화칸

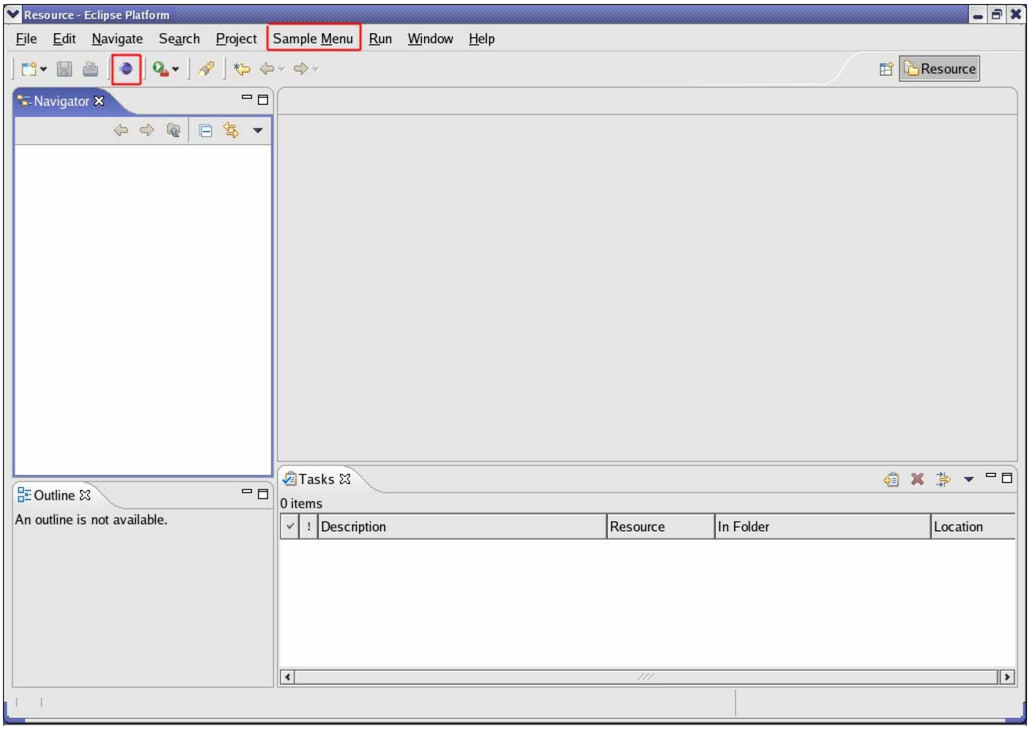


그림 13-8. 매몰된 Hello World끼워넣기

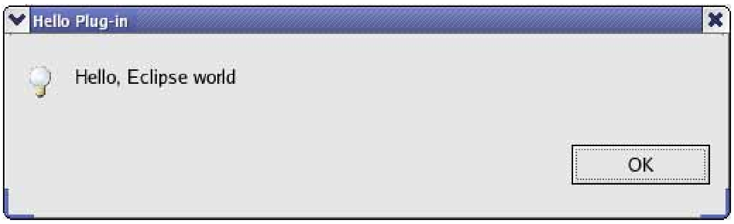


그림 13-9. Hello끼워넣기의 실행결과

Hello끼워넣기의 기능은
① 간단한 대화칸통보문을 현시하는것
② 차림표띠에 차림표를 추가하는것
③ 도구띠에 단추를 추가하는것이다.

우의 ①~③이 확인되면 끼워넣기를 실행한 Run-time작업대를 끈다.
끼워넣기를 작성할 때에 다음과 같은것을 진행해야 한다.

1. 끼워넣기를 Eclipse본체와 어떻게 통합하겠는가를 결정한다.
2. 끼워넣기를 통합하기 위하여 추가할 필요가 있는 확장점을 확인한다.
3. 확장점의 양상에 맞게 확장을 실현한다.
4. 제공하는 확장과 코드의 패키지를 서술하는 끼워넣기명시(plugin.xml)를 제공한다.

이로부터 먼저 실제로 끼워넣기명시나 실현클라스에 대하여 보기로 한다.

제3절. 끼워넣기명시

Hello끼워넣기를 만들 때 작업대에 《Welcome》이라고 쓴 편집기가 펼쳐져있었다. 이것이 끼워넣기명시이다. [Package Explorer]보기에서 plugin.xml을 연다. 그림 13-6에서 끼워넣기명시의 [Overview]페지를 보았다. [General Information]이라고 쓴 부분에 끼워넣기ID, 끼워넣기이름, 판번호, 제공자명, 클라스라는 정보가 있는것을 알수 있다. 동시에 [plugin.xml]페지를 보면 파일의 앞머리에 같은 정보가 씌여져있는 것을 알수 있다(목록 13-1). 이와 같이 끼워넣기명시에는 그것의 끼워넣기개발원천정보를 제일 앞에 서술한다.

목록 13-1. 끼워넣기명시의 원천코드 앞머리부분

```
<plugin
    id= “kp.ac.kut.example.hello”
    name= “Hello끼워넣기”
    version= “0.0.1”
    provider-name= “KUT”
    class= “kp.ac.kut.example.hello.HelloPlugin” >
```

다음 [Dependancies(의존관계)]페지를 열고 보면 [Required Plug-ins]에 《.org.eclipse.ui》와 《.org.eclipse.core.runtime》이 씌여져있다(그림 13-10). 여기서 오른쪽의 [Add]를 찰칵하여 [org.eclipse.core.resources]끼워넣기를 추가하여 둔다. 이 끼워넣기는 이 장의 뒤에서 보게 되는 끼워넣기전용화에서 필요하다. [plugin.xml]페지에서 보면 이러한 필수끼워넣기는 requires요소안에 서술된다(목록 13-2). requires요소에 쓸수 있는것은 이 끼워넣기가 반입하는 클라스를 포함하고있는 끼워넣기의 ID이다.

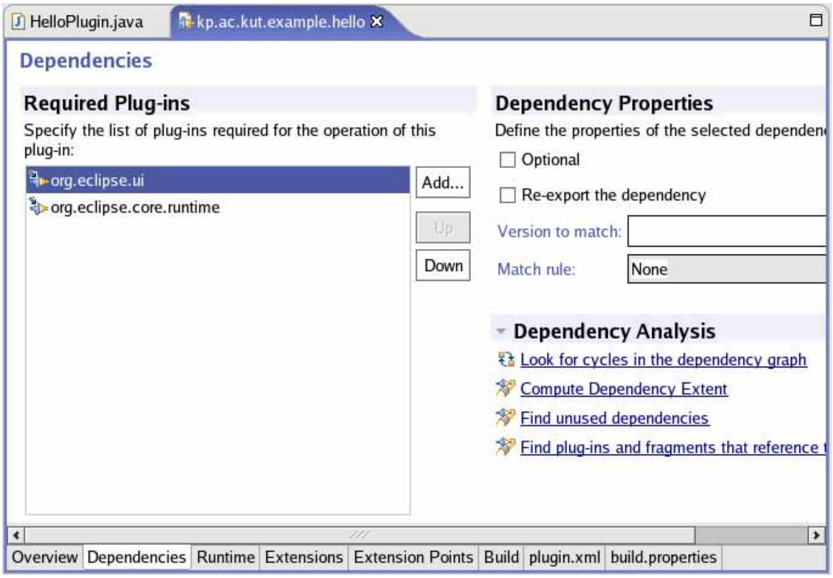


그림 13-10. Dependencies페지

목록 13-2. 끼워넣기명시원천프로그람의 require부분

```
<requires>
    <import plugin= “org.eclipse.ui” />
    <import plugin= “org.eclipse.core.runtime” />
    <import plugin= “org.eclipse.core.resources” />
</requires>
```

이번에는 [Run-time]페지를 열어본다(그림 13-11). [Run-time Libraries]라고 씌여진 곳에 hello.jar가 들어있다. Run-time Library는 이 끼워넣기의 클라스적재기의 유효범위를 결정한다. 지금은 hello.jar밖에 들어있지 않지만 필요에 따라 더 늘일수도 있다.

[Library exporting]은 서고를 다른 끼워넣기에서도 리용할수 있게 하겠는지를 지정한다. 확장점을 제공하고있지 않을 때에는 반출할 필요가 없다.

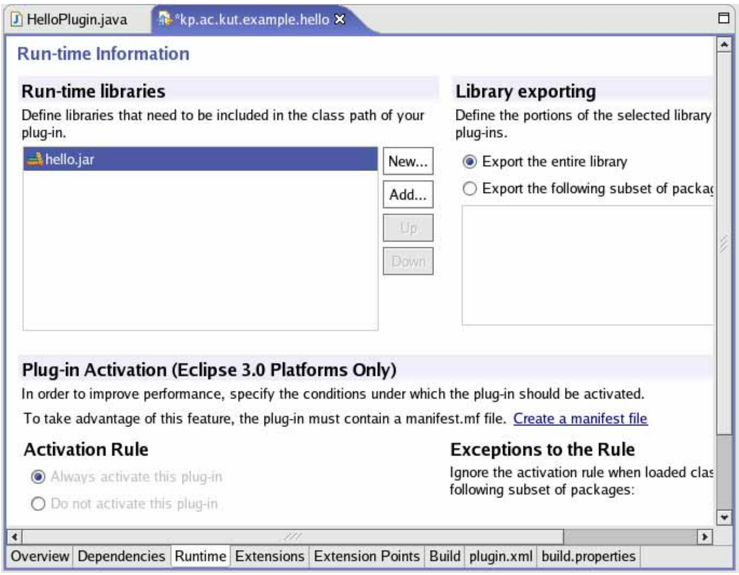


그림 13-11. Runtime페지

다음으로 [Extensions]페지를 보면 [All Extensions]에 《. org.eclipse.ui.actionSets》라고 되여있다. 이것은 이 끼워넣기가 확장한 확장점이다. 《org.eclipse.ui.actionSets》를 전개해나가면 [Sample Action]이라고 쓴것이 있을것이다(그림 13-12). 이 경로를 선택한다. 그러면 오른쪽의 령역의 [class]에 《kp.ac.kut.example.hello.actions.SampleAction》라고 되여있다. 확장점에 대하여는 [13.4 끼워넣기]에서 구체적으로 보기로 한다.

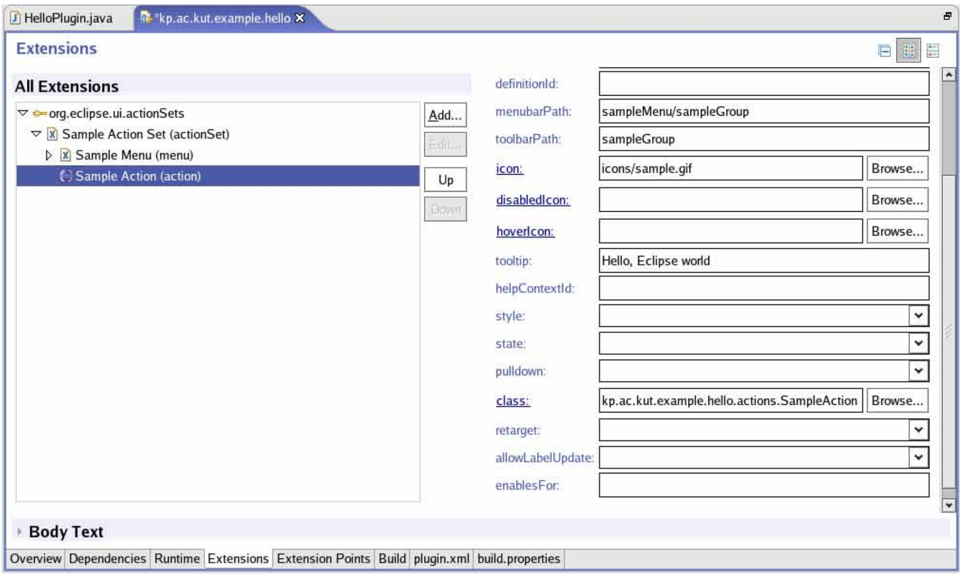


그림 13-12. Extension페지

끝으로 [Extension Points]페지이다. 여기에는 이 끼워넣기가 제공하는 확장점을 서술하지만 이번에는 사용하지 않기때문에 설명을 생략한다. 구체적인것은 도움말을 참조하기로 한다.

제4절. 끼워넣기

끼워넣기를 만들 때 제일 먼저 결정하여야 하는것은 자신의 끼워넣기와 Eclipse가 동기반을 어떻게 결합하겠는가 하는것이다. 즉 차림표띠에 차림표를 추가하는것과 도구띠에 단추를 추가하는 문제이다. 그것이 정해지면 다음에 확장점을 찾는다. 찾는 방법은 도움말에서 [Platform Plug-in Developer Guide]→[Programmer Guide]→[Plug-in to Workbench]→[Workbench's Extension Points]을 보는것이다. 그러면 그림 13-13에 끼워넣기가 진행된것을 볼수 있다. 즉 도구띠는 그림 13-13으로부터 [Action set actions]라는것을 알수 있다.

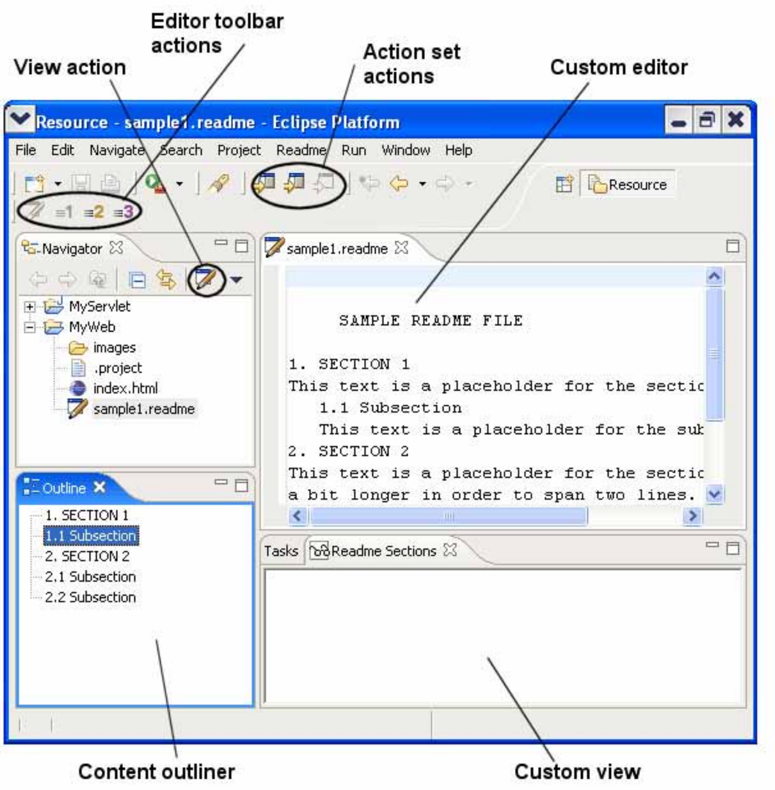


그림 13-13. 작업대의 기본확장점

이번에는 [Action set actions]에 대하여 구체적으로 살펴본다.

도움말의 [Platform Plug-in Developer Guide]→[Reference]→[Extension Points Reference]에는 확장점에 대한 설명이 있다. 끼워넣기는 이 방법에 따라 끼워넣어야 한다. 도움말의 [Extension Points Reference]를 선택하여 오른쪽판의 작업대항목의 [org.eclipse.ui.actionSets]를 읽어본다. 그러면 [API Reference]를 보면 끼워넣기를 끼워넣어야 할 대면이 [org.eclipse.ui.WorkbenchWindowActionDelegate]와 [org.eclipse.

ui.IworkbenchWindowPulldownDelegate]라는것을 알수 있다. 후자는 style 속성값이 《pulldown》인 경우에 끼워넣어야 한다(그림 13-14).

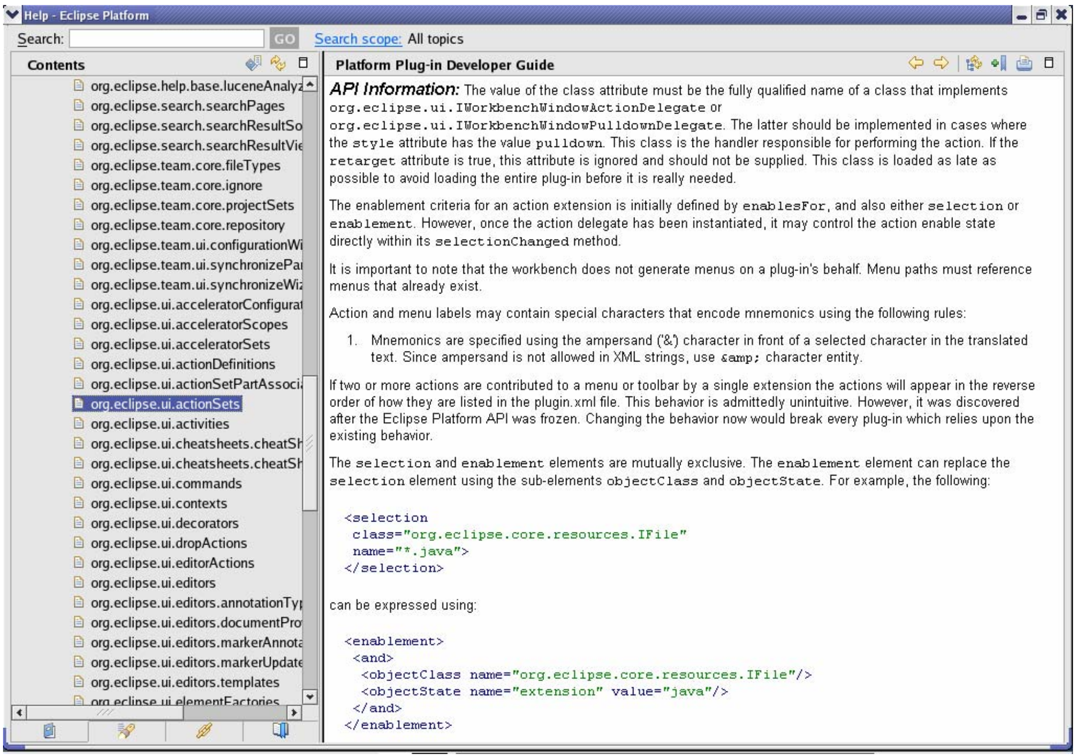


그림 13-14. 도움말의 API정보

Hello끼워넣기의 경우에 끼워넣기명시에서 확인해보면 style속성이 포함되여있지 않기때문에 전자를 끼워넣는것이 더 좋다(목록 13-3).

목록 13-3. 끼워넣기명시 원천코드의 extension요소

```
<extension point="org.eclipse.ui.actionSets">
    <actionSet
         label="Sample Action Set"
         visible="true"
         id="kp.ac.kut.example.hello.actionSet">
         <menu label="Sample &amp;Menu" id="sampleMenu">
            <separator name="sampleGroup">
            </separator>
         </menu>
         <action
              label="&amp;Sample Action"
              icon="icons/sample.gif"
              class="kp.ac.kut.example.hello.actions.SampleAction"
              tooltip="Hello, Eclipse world"
              menubarPath="sampleMenu/sampleGroup"
              toolbarPath="sampleGroup"
              id="kp.ac.kut.example.hello.actions.SampleAction">
         </action>
    </actionSet>
</additional room>
```

실제로 [kp.ac.kut.example.hello.actions.SampleAction]클라스를 편집기에서 열어보면 [IworkbenchWindowActionDelegate]대면을 실현하고있다는것을 알수 있다.

끼워넣기명시에는 어떠한 정보를 써넣어도 된다. 이미 본 도움말의 [org.eclipse.ui.actionSets]의 확장점참조서에 [Configuration Markup]이라는 항목이 있다. 여기에는 각 속성들이 무엇을 정의하고있는가를 설명하였다. 그러나 너무 많아서 어느것이 정말 필요한것인지 알아보기가 힘들다. 보통은 Hello끼워넣기와 같은 견본에서 사용하는 속성을 그대로 사용해도 문제는 없다. 그리고 도움말을 보면 사용방법을 리해할수 있을것이다.

여기서는 차림표띠에 차림표항목을 추가하고 도구띠에 단추를 추가하는 방법에 대해 약간 취급하려고 한다. 우선 Hello끼워넣기에서는 새로운 차림표항목을 작성한다. menu요소를 보면 label속성에 《SampleMenu》라고 씌여진 차림표항목이 정의되여있다(목록 13-4). id속성의 《sampleMenu》는 이 차림표의 ID이다. 도움말을 보면 알수 있듯이 path속성이 씌여져있지 않기때문에 이 sampleMenu는 새로 만드는 차림표로 정의된다.

목록 13-4. 끼워넣기명시 원천코드의 menu요소

```
<menu
      label="Sample &amp;Menu"
      id="sampleMenu">
   <separator
        name="sampleGroup">
   </separator>
</menu>
```

그 아래의 separator요소는 sampleMenu의 아래에 sampleGroup이라고 하는 분리기호로 구별되는 그룹을 만든다.

이번에는 action요소를 본다. menubarPath속성에 《sampleMenu/sampleGroup》이 적혀 있다(목록 13-5). 이것은 이 동작을 sampleGroup에 차림표항목으로 추가한다는것을 의미한다. Runtime작업대를 실행시키면 label속성값 [Sample Action]이 [Sample Menu]밑에 추가된다. 이렇게 차림표띠에 동작을 추가할 때에는 menuBarPath속성에 새로운 차림표의 경로를 입력한다.

마찬가지로 도구띠에 단추를 추가할 때에 toolbarPath속성에 단추를 추가하려는 도구띠그룹을 입력한다. Hello끼워넣기에서는 [SampleGroup]이라는 미정의 도구띠그룹을 지정하는것으로 도구띠에 새로운 그룹을 추가한다.

목록 13-5. 끼워넣기명시 원천코드의 action요소

```
<action
       label="&amp;Sample Action"
       icon="icons/sample.gif"
       class="kp.ac.kut.example.hello.actions.SampleAction"
       tooltip="Hello, Eclipse world"
       menubarPath="sampleMenu/sampleGroup"
       toolbarPath="sampleGroup"
       id="kp.ac.kut.example.hello.actions.SampleAction">
</action>
```

도구띠에 관계되는 속성에는 그 외에도 아이콘화상을 정의하는 icon속성과 도구암시를 표시하는 tooltip속성이 있다. icon속성에는 gif파일명이 plugin.xml파일로부터의 상대경로로 입력된다. 이 icon속성을 삭제해버리면 아이콘이 표시되지 않고 붉은 점으로 되여버린다.

끝으로 IWorkbenchWindowActionDelegate대면을 실현하여야 할 메쏘드에 대하여 본다.

. init()

이 끼워넣기가 초기화될 때 호출된다. 여기서는 부모웨인 window객체를 고속완충시켜 놓는다.

. selectionChanged()

작업대에서 선택되여 있던것이 달라지면 호출된다. 선택되여 있던것의 정보를 인수로 가지고 그것에 의해 동작을 비능동으로 할수 있다.

. run()

실제로 실행되는 동작을 서술한다.

. dispose()

완료될 때 호출된다. 자원 등의 후처리를 서술한다.

제5절. 끼워넣기의 전용화

계속하여 끼워넣기의 전용화를 진행한다. Hello끼워넣기의 《kp.ac.kut.example.hello.actions.SampleAction》클라스를 확장하여 선택된 파일이 XML일 때에만 Hello끼워넣기의 run()이 실행되게 해본다.

우선 SampleAction을 확장한 클라스를 만든다. [Package Explorer]안에서 [kp.ac.kut.example.hello.actions]패키지를 마우스오른쪽단추로 찰칵하고 [New]→[Class]를 선택한다. 그러면 [New Java Class]조수가 기동하는데 이름은 [MyAction], 상위클라스는 [SampleAction]으로 한다(그림 13-15). 그다음 [Finish]를 찰칵한다.

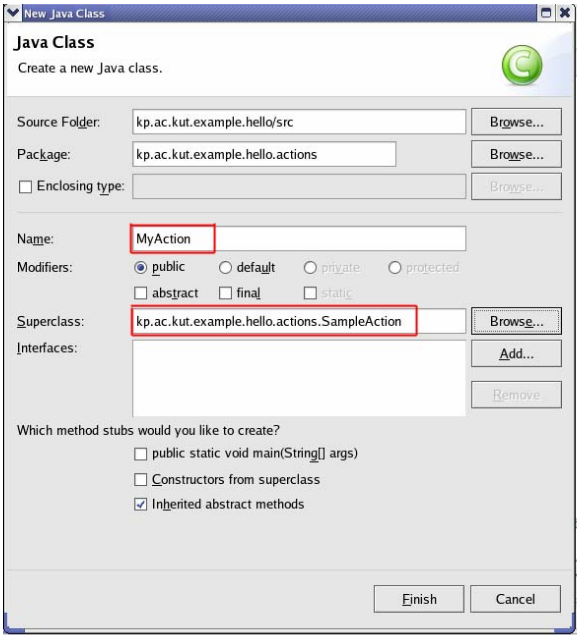


그림 13-15. 새로운 Java클라스작성대화칸

다음으로 끼워넣기명시의 action요소안의 class속성과 id속성을 변경한다. 지금까지는 [SampleAction]이였지만 이번에는 [MyAction]으로 한다(목록 13-6).

목록 13-6. 끼워넣기명시 원천코드의 action요소(변경 후)

```
<action
        label="&amp;Sample Action"
        icon="icons/sample.gif"
        class="kp.ac.kut.example.hello.actions.MyAction"
        tooltip="Hello, Eclipse world"
        menubarPath="sampleMenu/sampleGroup"
        toolbarPath="sampleGroup"
        id="kp.ac.kut.example.hello.actions.MyAction">
</action>
```

MyAction클라스를 편집기에서 열어 편집한다.

먼저 SampleAction의 메쏘드를 재정의한다. 차림표띠에서 [source]→[Override/Implement Methods]를 선택하여 현시된 대화칸에서 재정의하는 메쏘드를 선택하고 [OK]를 누르면 자동적으로 재정의할수 있다(그림 13-16, 목록 13-7). 또한 취미에 따라 메쏘드설명문도 추가할수 있다.

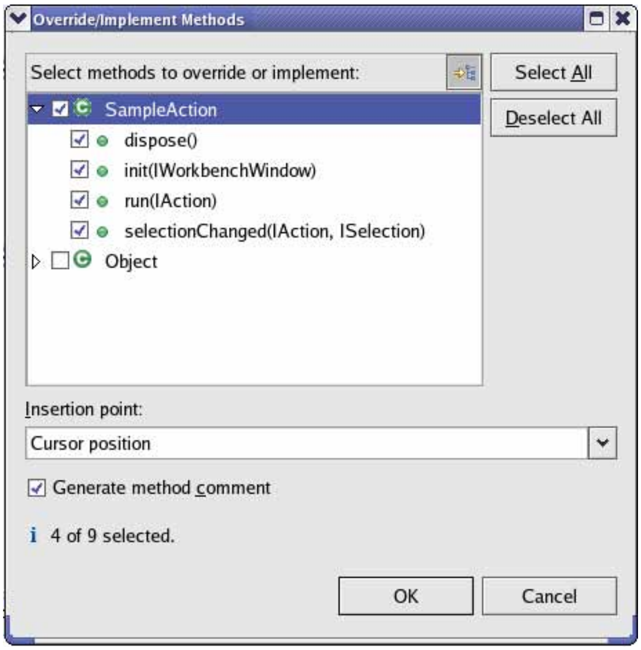


그림 13-16. Override/Implement Methods대화칸

목록 13-7. 재정의된 메쏘드

```
public void dispose() {
   super.dispose();
}
public void init(IWorkbenchWindow window) {
   super.init(window);
}
public void run(IAction action) {
   super.run(action);
}
public void selectionChanged(IAction action, ISelection selection) {
   super.selectionChanged(action, selection);
}
```

그러면 재정의한 메쏘드를 확장해본다.

처음으로 선택된 파일을 취하기 위하여 selection객체를 정의한다. 여기서 selectionChanged()가 실행될 때마다 selection객체를 갱신되게 한다.

```
private ISelection selection;
public void selectionChanged(IAction action, ISelection selection) {
    this.selection=selection;
    super.selectionChanged(action, selection);
}
```

이번에는 실행되는 동작을 서술하는 run메쏘드를 작성한다. 처음에는 파일이 선택되여있는지를 확인한다. 그 방법으로 selection객체의 구체례가 IstructuredSelection과 일치하는가를 조사한다. 만일 일치하지 않으면 되돌리기하여 동작을 실행하지 않는다.

```
if(this.selection == null || this.selection.isEmpty()
    || !(this.selection instanceof IStructuredSelection)){
return;
}else{
//do nothing
}
```

다음에 파일을 취한다. 코드를 보면 알수 있듯이 IStructuredSelection안에 IFile을 취한다. 이미 전에 필요한 끼워넣기의 추가에서 [org.eclipse.core.resources]를 추가하였는데 이 끼워넣기는 IFile 등 자원계렬의 처리를 진행할 때 필요하다.

```
for(Iterator itr = ((IStructuredSelection)this.selection).iterator();
        itr.hasNext();
        ){
          Object target = itr.next();
          if(target instanceof IFile){
             IFile file = (IFile)target;
```

얻어 낸 파일을 검사하기 위한 메소드를 정의한다.

```
private void variable(IFile file) throws Exception {
        InputStream is = file.getContents();
        byte[] fileheader = new byte[5];
        byte[] xmlheader = { '<', '?', 'x', 'm', 'l'}
        is.read(fileheader);
        is.close();
        for(int i = 0; i < 5; i++){
            if( !(xmlheader[i] == fileheader[i])) {
             throw new java.lang.Exception();
            }
        }
}
```

이상의 내용을 종합하면 목록 13-8과 같다.

목록 13-8. 확장한 메쏘드

```
  public void run(IAction action) {
     if(this.selection == null || this.selection.isEmpty()
             || !(this.selection instanceof IStructuredSelection) ) {
        return ;
     } else { //do nothing
     }
     for(Iterator itr =((IStructuredSelection)this.selection).iterator();
               Itr.hasNext();) {
        Object target = itr.next();
        if(target instanceof IFile) {
           IFile file = (IFile)target;
           try {
             validate(file);
           } catch (CoreException e) {
             e.printStackTrace();
             return;
           } catch(IOException e) {
             e.printStackTrace();
             return;
           } catch(Exception e){
             System.out.println(file.getName()+"는 XML파일이 아닙니다.");
             return;
           }
        } else {
           return;
```

```
        }
      }
      System.out.println("선택한 파일은 모두 XML파일입니다.");
      super.run(action);
   }
   private void validate(IFile file) throws Exception {
      byte[] fileHeader = new byte[5];
      byte[] xmlHeader = {'<', '?', 'x', 'm', 'l'};

      InputStream is = null;
      is = file.getContents();
      is.read(fileHeader);
      is.close();
      for(int i=0 ; i < 5 ; i++) {
        if(!(xmlHeader[i] == fileHeader[i])) {
           throw new java.lang.Exception;
        }
      }
   }
```

편집한 결과를 보존하고 Runtime작업대를 실행해본다(실행하기 전에 주의할것은 끼워넣기명시의 [Dependencies]페지에서 [Required Plug-ins]에 [org.eclipse.core.resources]가 포함되여있는지를 확인하는것이다.). 그러면 《…cannot be resolved or is not a type》라는 오유가 나올수 있는데 오유행왼쪽에 표시되여있는 아이콘을 찰칵하여 필요한 패키지를 반입한다.

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Iterator;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IWorkbenchWindow;
```

Runtime작업대가 열리면 Navigator보기에서 프로젝트를 만들고 그 안에 아무것도 써넣지 않은 빈 파일을 작성하여 시험해본다. 빈 파일file.txt을 선택하여 [Hello, Eclipse world]단추를 눌러 본다. 아무런 동작도 진행되지 않고 그림 13-17과 같이 끼워넣기개발투시의 조종탁에 《file.txt는 XML파일이 아닙니다.》가 출력된다.



그림 13-17. 조종탁에 통보문이 현시

이번엔 .project파일을 선택하여 [Hello Eclipse world]단추를 눌러 본다. 그러면 그림 13-18과 같이 [Hello끼워넣기]대화칸이 나타난다. 이때 끼워넣기개발투시의 조종탁에는 《선택한 파일은 모두 XML파일입니다.》가 출력된다(그림 13-19).

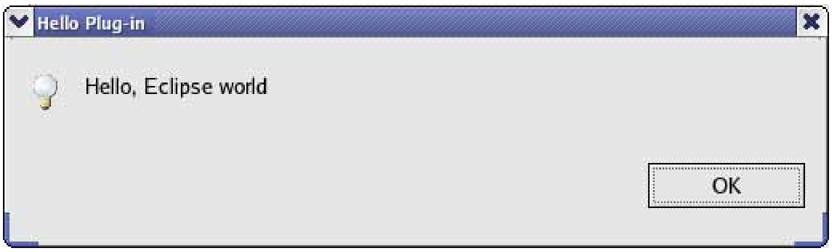


그림 13-18. [Hello끼워넣기]대화칸

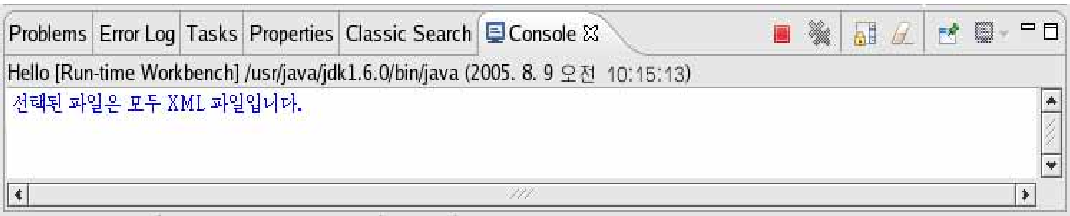


그림 13-19. 조종탁에도 통보문이 현시

제6절. 끼워넣기의 반출

끼워넣기의 반출은 단순하다. 끼워넣기프로젝트를 선택하고 차림표에서 [Export]을 실행하면 된다. 구체적으로 도움말의 [PDE Guide]→[Getting Started]→[Basic Plug-in Tutorial]→[Deploying a plug-in]를 참조하면 된다. 여기서는 간단히 순서를 설명한다.

끼워넣기프로젝트를 선택하고 마우스오른쪽단추를 찰칵하여 현시되는 차림표에서 [Export]을 선택한다. 현시되는 대화칸에서 [Deployable plug-ins and fragments]을 선택하고 [Next]를 찰칵한다.

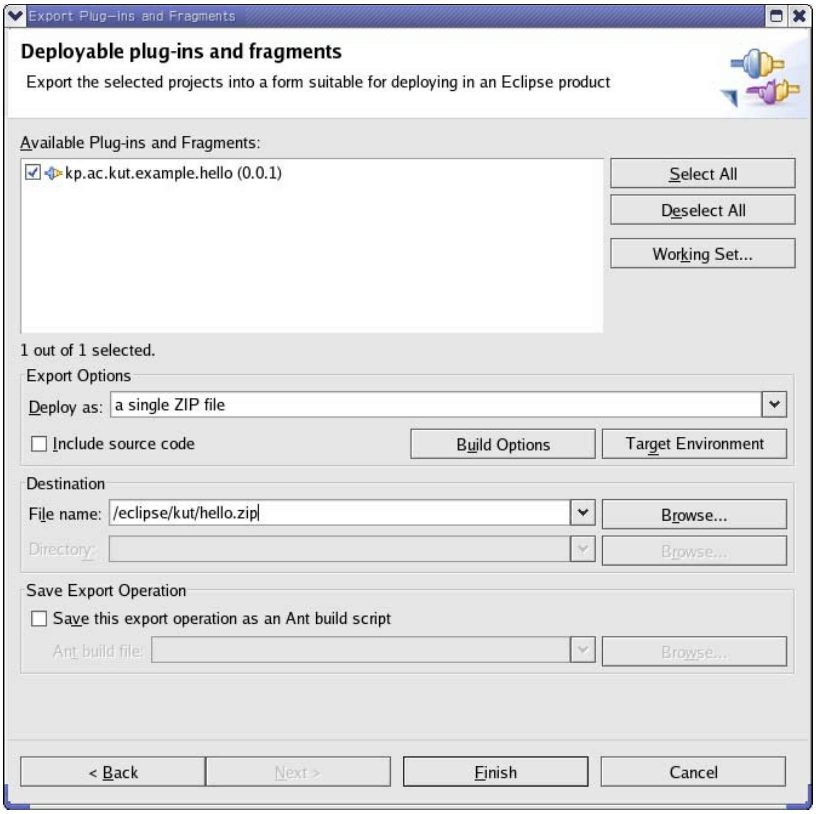


그림 13-20. 반출대화칸

[Available Plug-ins and Fragments]에서 [kp.ac.kut.example.hello]를 선택하고 [Export Option]에서 [a single ZIP file]을 선택한다. 그리고 [File name]에 [/eclipse/kut/hello.zip]를 입력한 다음 [Finish]를 찰칵한다(그림 13-20).

제14장. Eclipse RCP

제1절. Eclipse RCP의 개요

Eclipse 3.0에서는 RCP(Rich Client Platform:다기능의뢰기가동기반)기능이 강화되였다. Eclipse RCP는 Eclipse를 통합개발환경(IDE)의 가동기반으로 리용하던것으로부터 공통의 응용프로그람가동기반(응용프로그람을 구축할 때 사용하는 기반)으로 확대하겠다는 시도에서 나온것이다.

Eclipse는 현재도 응용프로그람가동기반으로 리용할수 있지만 응용프로그람가동기반으로만 동작시키는데 필요없는 기능이 적지 않다. 실례로 JDT(Java Development Tool:Java개발도구)이다. 명백히 개발에는 JDT가 필요하지만 사용자가 리용하는 응용프로그람에서는 필요가 없다. 또한 IDE에는 필수적인 본문편집기능도 사용자에게는 필요없는 기능으로 밖에 되지 않는다.

여기서 《필요없는 기능을 빼고 Eclipse를 기동시키면 좋지 않은가》하는 착상이 생긴다. 그러나 Eclipse 2.x까지의 판본에서는 IDE가동기반으로 설계되여있으므로 다기능의뢰기응용프로그람을 실현하는 기능만을 떼여 내기가 곤난하다. 그림 14-1을 본다. UI부품부분에 IDE가동기반으로 하여 필요한 Text(본문편집기 등 Eclipse에서 제공하는 편집기의 조종부분)와 IDE의 보기부분이 꽉 차있어 Text부품과 IDE View부품을 빼낸다는것이 힘들다.

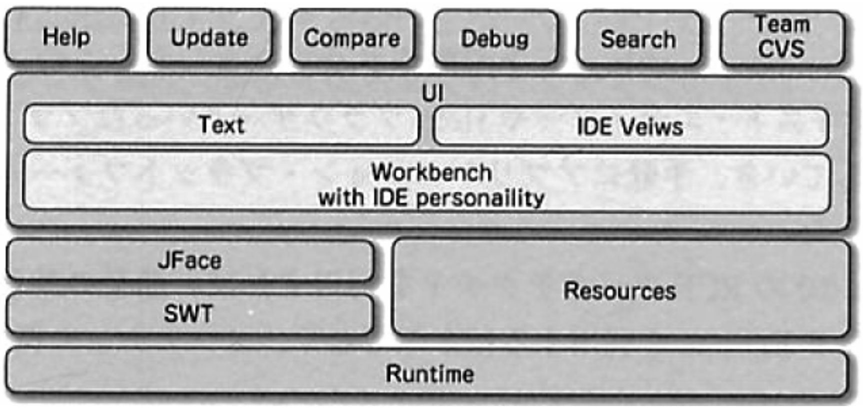


그림 14-1. Eclipse 2.x까지의 구성방식

이렇게 Eclipse 2.x판본까지의 결함을 퇴치하기 위해 Eclipse 3.0에서는 다기능의뢰기응용프로그람의 구축이 헐하게 구성방식이 변경되였다. Eclipse 3.0의 구성방식은 그림 14-2와 같다.

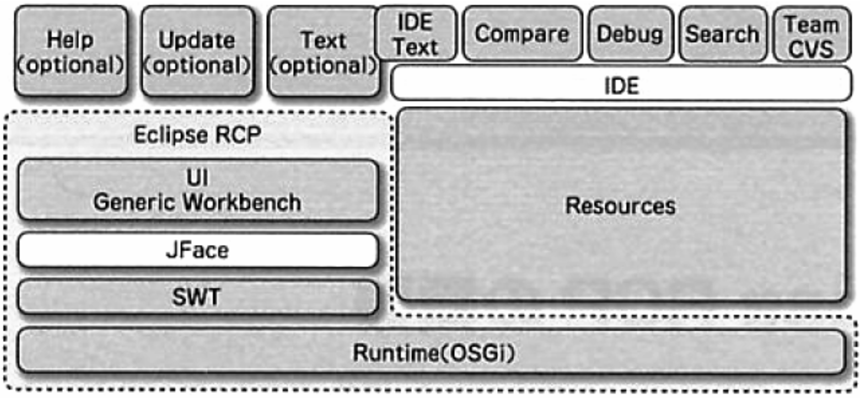


그림 14-2. Eclipse 3.0의 새로운 구성방식

그림 14-2를 보면 알수 있듯이 그림 14-1에서 UI부품에 들어가 있던 Text부품과 IDE View부품들이 서로 분리되였다. UI부품은 응용프로그람의 가동기반으로 리용하기 쉽게 범용적으로 만들었다. 결국 Eclipse의 작업대로부터 IDE에 고유한 필요한 부분품의 의존을 없애고 일반적인 응용프로그람에서도 리용가능하게 되였다. 결국 범용적인 작업대로 된것뿐이다.

Eclipse RCP를 리용한 응용프로그람을 기동하는데는 Generic Workbench와 JFace, SWT, Runtime(OSGi)가 있으면 된다. 도움말이나 본문비교, 본문편집기의 기능 등 Eclipse부속의 끼워넣기기능을 리용하고싶을 때에는 끼워넣기기능을 더 추가하면 된다.

기동에 필요한 부분품들이 아주 적어졌으므로 보통의 Eclipse와 비교하면 기동시간이 짧고 응용프로그람자체의 동작도 가벼워 졌다. 이 장에서는 실제로 Eclipse RCP를 리용한 응용프로그람에 대하여도 설명하기때문에 보통의 Eclipse와 속도를 비교할수 있다.

이 새로운 Eclipse 3.0의 RCP구성방식을 리용하면 보통의 업무응용프로그람의 가동기반으로 Eclipse를 리용할뿐 아니라 개발환경으로도 쓸수 있다.

실례로 통신쎈터의 응용프로그람을 Eclipse RCP로 구축하였다고 하면 운영중에 고장이 생긴 경우 Eclipse RCP응용프로그람에 끼워넣기기구를 리용하여 오유수정환경을 만들고 원인을 찾아내여 문제를 처리할수 있다. Eclipse의 다음 판본에서는 완전히 동적으로 끼워넣기를 추가할수 있게 할 예정이다. 이 기능을 리용하면 응용프로그람을 재기동하지 않고 실례로 본 오유수정환경을 구축할수 있다.

제2절. Eclipse RCP

RCP라고 해도 그렇게 힘들기만 한것은 아니다. 끼워넣기개발때 소개한 끼워넣기개발순서와 거의 같다. 그래도 약간의 차이가 있으므로 주의를 돌리면서 RCP개발순서를 따라 해보기로 한다.

14.2.1. HelloRCP응용프로그람의 개요와 준비

제일 먼저 만드는 견본은 [HelloWorld]프로그람이다. 그러나 이번 견본은 단순히 빈(아무런 보기도 편집기도 표시되지 않는) 작업대를 띄우는것뿐이며 《HelloWorld》조차 현시하지 않는다. 그것은 Eclipse RCP를 리용하는 응용프로그람(RCP응용프로그람)을 작성하는데 맞게 최소한의 필요한 구성밖에 리용하지 않기때문이다. 그 최소한의 구성만으로 응용프로그람을 기동해보면서 Eclipse RCP의 기본을 학습해본다.

먼저 [Plug-in Development]투시를 열어본다. 차림표띠의 [Window]→[Open Perspective]→[Other]를 선택하면 [Select Perspective]대화칸이 열린다. 그 안에서 [Plug-in Development]을 선택하고 [OK]를 찰칵하면 [Plug-in Development]투시가 열린다.

14.2.2. 끼워넣기프로젝트조수의 사용

Eclipse RCP를 리용한 응용프로그람을 만들어본다. [Package Explorer]보기에서 마우스오른쪽단추를 찰칵하고 [New]→[Plug-in Project]를 선택하면 [New Plug-in Project]조수가 기동한다. 제일 먼저 나오는 [Plug-in Project]페지(그림 14-3)에서 프로젝트이름은 《SampleChap14.1》로 하고 [Use default]와 [Create a Java project]선택칸을 선택한다.

[Alternate Format(For Advanced Users Only):형식변경(전문가용)]그룹안에 있는 [Create an OSGi bundle manifest for the plug-in(끼워넣기를 위한 OSGi묶음명시를 생성)]칸은 선택하지 않는다. 이 선택항목은 Eclipse 3.0부터 생긴것인데 끼워넣기를 OSGi대응끼워넣기로 하기 위한 《Manifest.MF》라고하는 파일을 작성할 때 리용한다. 다음 [Next]를 찰칵한다. 그러면 [Plug-in Content]대화칸이 나타난다 (그림 14-4). [Plug-in Properties]그룹안에 다음과 같은 값들을 넣는다. 끼워넣기ID에는 《sample.hello》, 끼워넣기판번호에는《0.01》, 끼워넣기이름은 《HelloRcp》으로 한다. 끼워넣기제공자에는《KUT》를 입력한다. [Runtime Library]에는《HelloRcp.jar》라고 입력한다. 또한 [Plug-in Class]그룹에 있는 [Generate the Java class that controls the pulg-in's life cycle]칸을 이번 RCP응용프로그람에서는 사용하지 않으므로 선택하지 않는다. 그리고 형판은 사용하지 않으므로 조수에서의 설정은 여기서 끝낸다. [Finish]를 찰칵한다

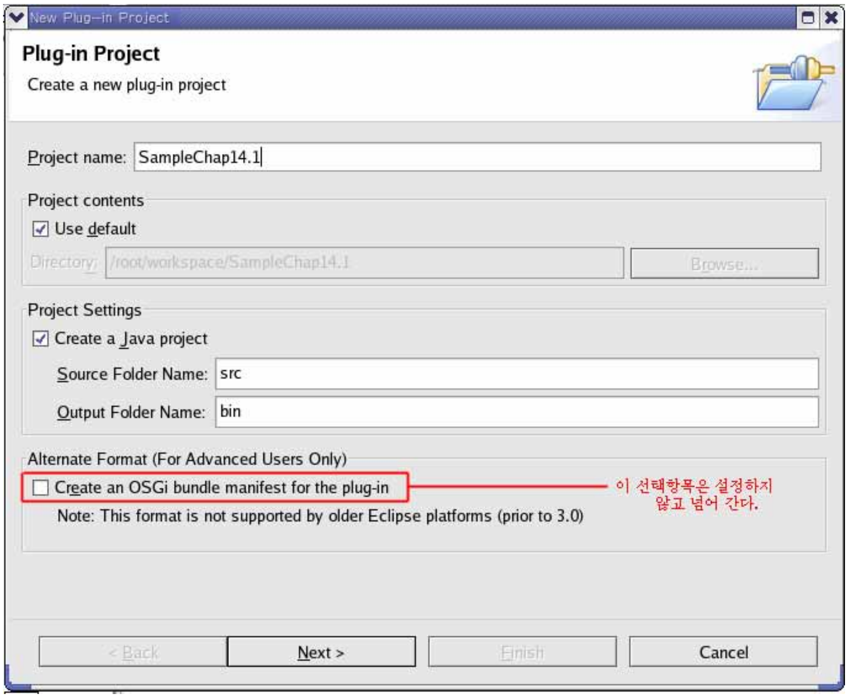


그림 14-3. 끼워넣기프로젝트작성대화칸

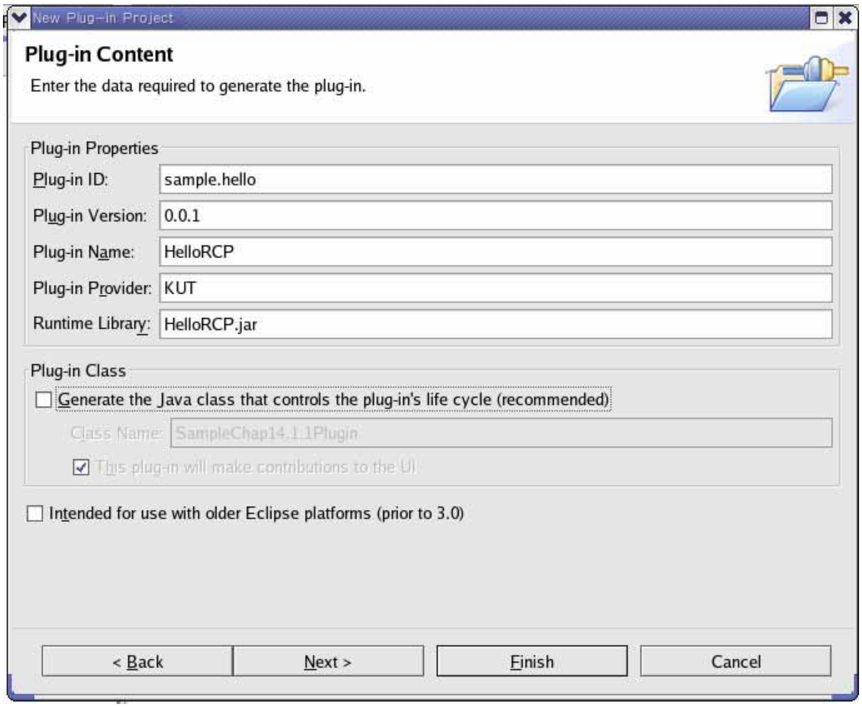


그림 14-4. 끼워넣기내용대화칸

14.2.3. 필수끼워넣기의 추가

다음으로 RCP응용프로그람에 필요한 끼워넣기를 추가한다. [Package Explorer]보기에 있는 [plugin.xml]을 두번 찰칵하여 열고 [Dependancies]페지의 [Required Plug-ins(필수끼워넣기)]령역에 있는 [Add]를 누르고 《org.eclipse.core.runtime》과 《org.eclipse.ui》를 추가한다(그림 14-5).

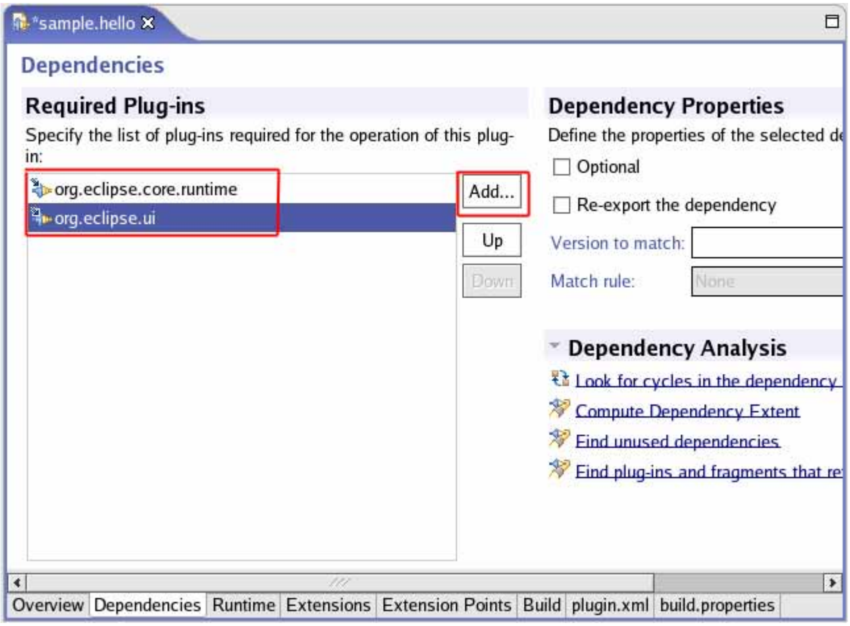


그림 14-5. Dependancies페지

14.2.4. RCP응용프로그람에 필요한 클라스의 정의

끼워넣기명시에 필요사항을 모두 서술하기 전에 먼저 클라스를 만든다. 그것은 끼워넣기명시의 서술을 계속해 나가자면 클라스지정이 필요하기때문이다. RCP응용프로그람을 작성하기 위하여 최소로 필요한 클라스는 다음의 3개이다.

① IPlatformRunnable대면의 실현클라스

IPlatformRunnable대면의 실현클라스는 응용프로그람의 입구점으로 되는 클라스이다. 그것은 IPlatformRunnable대면의 실현클라스의 run메쏘드로부터 RCP응용프로그람이 시작되기때문이다. 말하자면 run메쏘드는 보통의 Java클라스에서 main메쏘드와 같은것이다.

② WorkbenchAdvisor추상클라스의 하위클라스

WorkbenchAdvisor추상클라스의 하위클라스는 다기능의뢰기(Rich client)의 기반으로 되는 작업대의 설정을 진행하는 클라스이다. RCP응용프로그람은 반드시 투시가 한

개 이상 필요하다. 이 투시를 호출하는 역할을 WorkbenchAdvisor추상클라스의 하위클라스가 하는것이다. 그래서 이 클라스는 응용프로그람의 수명을 관리하는 역할을 한다. 결국 RCP응용프로그람동작중의 화면에 관계되는 사건이 일어날 때마다 기동되는 메쏘드가 정의된다.

구체적인것은 이 장의 3절에서 설명한다.

③ IPerspectiveFactory대면의 실현클라스

RCP응용프로그람은 반드시 한개 이상의 투시가 필요하다는것을 우에서 설명하였다. 바로 IPerspectiveFactory대면의 실현클라스가 그 투시를 만드는 역할을 하는 클라스이다.

그러면 차례로 실제클라스들을 작성하여 보기로 한다. 아래에 정의하는 클라스들은 [Package Explorer]보기의 [SampleChap14.1]프로젝트의 [sample.hello]패키지에 작성한다.

HelloRcpApplication클라스(IPlatformRunnable대면의 실현클라스)

목록 14-1. HelloRcpApplication클라스

```
package sample.hello;

import org.eclipse.core.runtime.IPlatformRunnable;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.application.WorkbenchAdvisor;

public class HelloRcpApplication implements IPlatformRunnable {
    public Object run(Object args) {
        WorkbenchAdvisor workbenchAdvisor = new HelloRcpAdvisor();
        Display display = PlatformUI.createDisplay();
        try {
            int returnCode = PlatformUI.createAndRunWorkbench
                        (display, workbenchAdvisor);
            if (returnCode == PlatformUI.RETURN_RESTART) {
                return IPlatformRunnable.EXIT_RESTART;
            } else {
                return IPlatformRunnable.EXIT_OK;
            }
        } finally {
        display.dispose();
        }
    }
}
```

RCP응용프로그람의 기동점을 정의하는 클라스에서는IPlatformRunnable대면의 run메쏘드를 적용할 필요가 있다. 그 run메쏘드를 적용한것이 목록 14-1이다. run메쏘드안에서는 제일 처음 WorkbenchAdvisor객체의 구체례를 생성한다. 다음 PlatformUI클라스의 createDisplay메쏘드를 사용하여 Display객체를 생성한다.

이 2개의 workbenchAdvisor구체례와 display구체례를 리용하여 PlatformUI클라스의 createAndRunWorkbench메쏘드로 작업대를 기동한다. 작업대가 어떠한 리유로 완료될 때까지 createAndRunWorkbench메쏘드는 계속 실행되여있다. 어떠한 리유로 작업대가 완료된 경우 createAndRunWorkbench메쏘드는 아래의 값들중 하나를 귀환값으로 돌린다.

PlatformUI.RETURN_OK 보통완료

PlatformUI.RETURN_RESTART 재기동요구에 의한 완료

PlatformUI.RETURN_UNSTARTABLE 작업대개시실패

PlatformUI.RETURN_EMERGENCY_CLOSE 이상완료

이러한 정수들은 Java편집기상의 출현위치에서 Ctrl건을 누르면서 마우스지적자를 정수에 대고 찰칵하면 정의되여 있는 부분에로 이행할수 있는것들이다. 이행한 정의부에 설명문이 있으므로 한번 읽어본다.

run메쏘드의 귀환값은 IPlatformRunnable대면에 정의된 정수 EXIT_OK, EXIT_RESTART, EXIT_RELAUNCH 중의 하나이다. 우의 코드에는 createAndRunWorkbench메쏘드의 귀환값이 PlatformUI.RETURN_RESTART인 경우에 IPlatformRunnable.EXIT_RESTART를, 그렇지 않은 경우에는 IPlatformRunnable.EXIT_OK를 run메쏘드의 귀환값으로 되돌린다.

작업대가 완료되여도 SWT의 Display객체가 처리되는것이 아니기때문에 반드시 Display객체의 dispose메쏘드를 호출한다. 이 Display클라스는 SWT에 정의되여있는 클라스로서 화면에는 아무것도 표시하지 않지만 구체례가 생성되면 조작체계로부터 여러가지 GUI자원을 획득한다. display객체의 dispose()메쏘드를 호출하여 자원을 해방하는것이 SWT에서 류의할 점이다.

HelloRcpAdvisor클라스(WorkbenchAdvisor추상클라스의 하위클라스)

WorkbenchAdvisor클라스의 하위클라스는 getInitialWindowPerspectiveId메쏘드를 적용할 필요가 있다. 아래의 getInitialWindowPerspectiveId메쏘드(목록 14-2)에서는 작업대에 필요한 투시의 ID문자렬을 되돌린다.

목록 14-2. HelloRcpAdvisor클라스

```
package sample.hello;

import org.eclipse.ui.application.WorkbenchAdvisor;

public class HelloRcpAdvisor extends WorkbenchAdvisor {
    public String getInitialWindowPerspectiveId() {
        return "sample.hello.HelloRcpPerspective";
    }
}
```

HelloRcpPerspective클라스(IPerspectiveFactory대면의 실현클라스)

IPerspectiveFactory대면을 실현한 클라스는 createInitialLayout메쏘드를 적용할 필요가 있다. 보통은 이 메쏘드안에서 보기나 편집기 등의 작성을 하지만 이번에는 최소판임으로 아무것도 처리하지 않는다(목록 14-3).

목록 14-3. HelloRcpPerspective클라스의 선언

```
package sample.hello;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

public class HelloRcpPerspective implements IPerspectiveFactory {
   public HelloRcpPerspective(){
   /* default constructor */
   }
   public void createInitialLayout(IPageLayout layout) {
   }
}
```

이상 클라스의 작성을 끝냈다. 실제로 견본응용프로그람의 개요에서 설명한것처럼 단순히 빈 작업대를 기동하기 위한 응용프로그람이지만 그렇게 간단하지는 않다.

14.2.5. 끼워넣기명시에서 투시정의

Eclipse의 다기능의뢰기응용프로그람에서는 최소 한개이상의 투시가 필요하다. 투시를 정의하기 위해 확장점[org.eclipse.ui.perspectives]를 확장한다.

[Package Explorer]보기에서 [SampleChap14.1]프로젝트의 plugin.xml을 두번 찰칵하여 다시 끼워넣기명시를 열고 [Extensions]페지를 펼친다. [Add]를 찰칵하여 확

장점[org.eclipse.ui.perspectives]를 추가한다. 그다음 [All Extensions]령역안의 [org.eclipse.ui.perspectives]에서 마우스오른쪽단추를 찰칵하여 [New]→[perspective]를 선택한다. 그러면 오른쪽에 [Extension Element Details]구역이 나타난다.

[Extension Element Details]안에서 "Perspective"의 속성들인 name, class, id를 설정한다. 처음에 투시를 정의하는 클라스를 [Browse]단추를 찰칵하여 선택한다. 이미 작성한 HelloRcpPerspective클라스를 선택한다. 선택하면 HelloRcpPerspective클라스의 완전한 이름이 표시된다. 이 이름을 그대로 id값으로 하여 사용한다. name에는 《HelloRcpPerspective》라고 입력한다(그림 14-6).

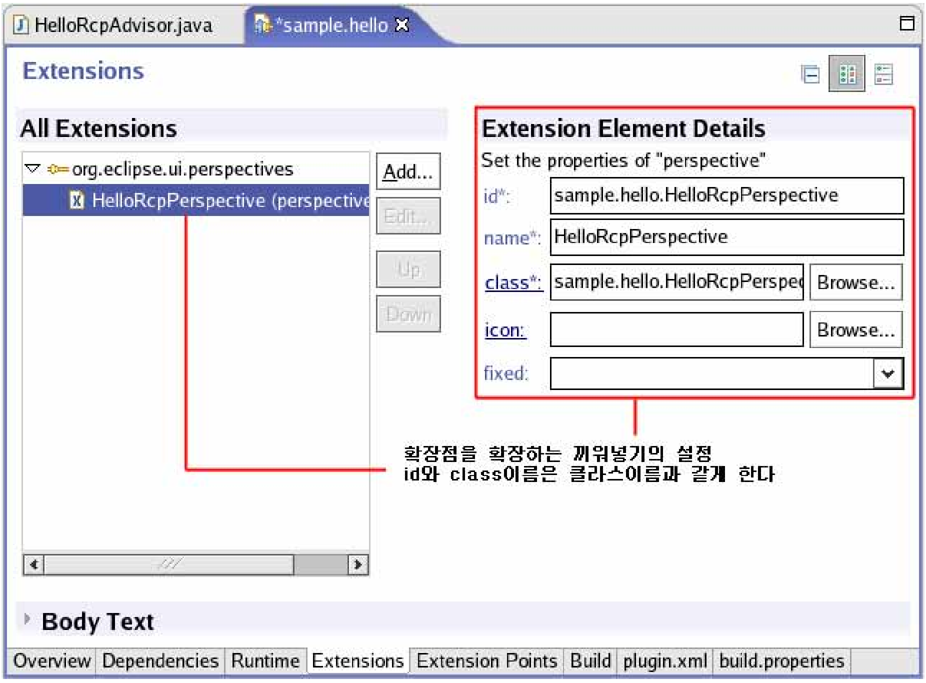


그림 14-6. 끼워넣기명시에서 투시정의

14.2.6. 끼워넣기명시에서 응용프로그람의 정의

끼워넣기명시에서 응용프로그람의 정의를 진행한다. 응용프로그람의 정의는 [org.eclipse.core.runtime.applications]확장점을 확장하여 작성한다.

[Extensions]페지에서 [All Extensions]의 [Add]를 찰칵하여 《org.eclipse.core.runtime.applications》를 추가한다. 거기서 [Extension Details]의 ID에 《HelloRcpApplication》을 입력한다. 여기서 입력하지 않으면 RCP응용프로그람을 실행할수 없으므로 반드시 입력한다(그림 14-7).

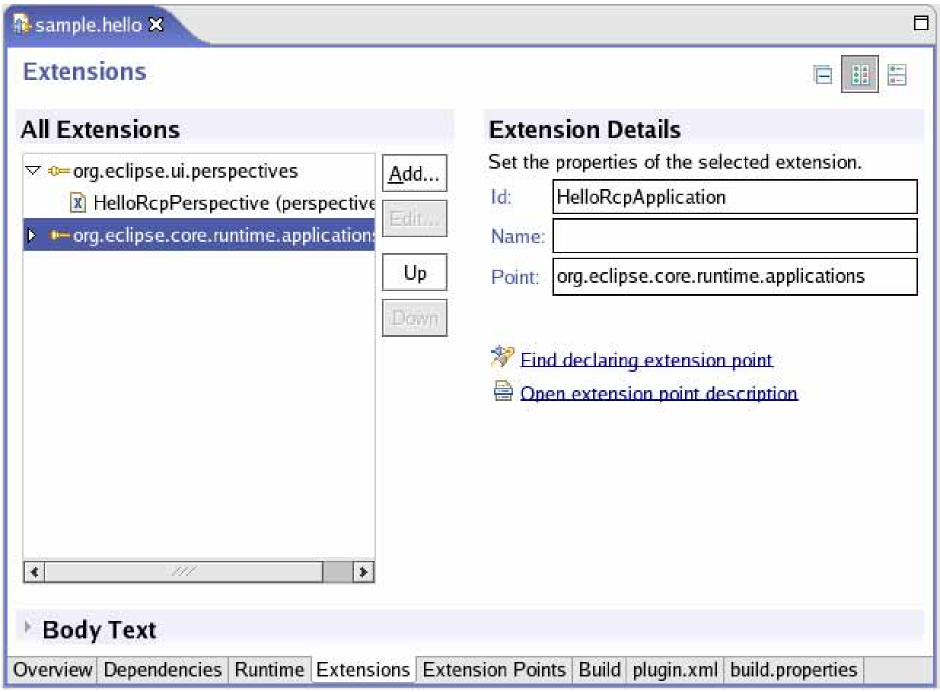


그림 14-7. 끼워넣기명시에서 응용프로그람의 정의

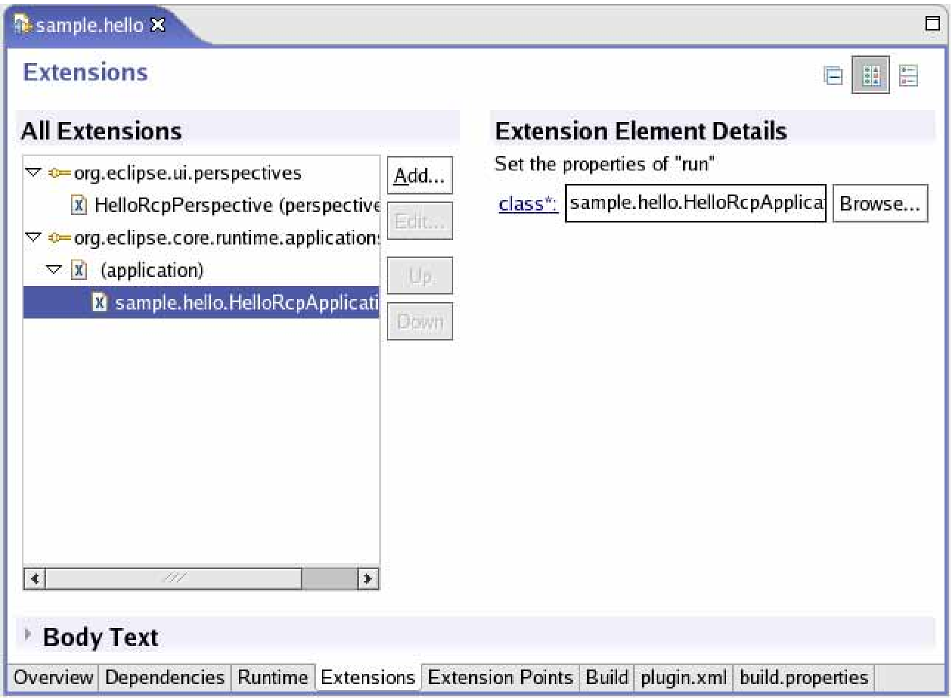


그림 14-8. 끼워넣기명시에서 응용프로그람의 정의

다음 [All Extensions]에서 [org.eclipse.core.runtime.applications]를 마우스 오른쪽단추로 찰칵하고 [New]→[application]을 선택한다. 다음 [application]을 오른쪽찰칵하고 [New]→[Run]을 선택한다. 그러면 [Run]요소의 [Extension Element Details]가 표시되는데 먼저 run메쏘드를 정의한 클라스 [sample.hello.HelloRcpApplication]을 선택한다(그림 14-8). 이 클라스는 IPlatformRunnable대면을 실현하고있다.

이상 RCP응용프로그람을 동작시키는 최소한의 끼워넣기명시 정의가 끝났다. 끝으로 [plugin.xml]페지를 본다. 이 페지에서는 끼워넣기명시의 원천코드를 볼수 있다(목록 14-4).

목록 14-4. HelloRcp응용프로그람의 끼워넣기명시

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="sample.hello"
   name="HelloRcp"
   version="0.01"
   provider-name="Rich Client Task %providerName">

 <runtime>
      <library name="HelloRcp.jar">
           <export name="*"/>
      </library>
 </runtime>
 <requires>
      <import plugin="org.eclipse.core.runtime"/>
      <import plugin="org.eclipse.ui"/>
 </requires>
 <extension
      id="HelloRcpApplication"
      point="org.eclipse.core.runtime.applications">
      <application>
        <run class="sample.hello.HelloRcpApplication"/>
      </application>
 </extension>
 <extension
      point="org.eclipse.ui.perspectives">
      <perspective
          class="sample.hello.HelloRcpPerspective"
          name="HelloRcpPerspective"
          id="sample.hello.HelloRcpPerspective"/>
 </extension>
</plugin>
```

14.2.7. RCP응용프로그람실행

견본응용프로그람을 실행시켜 본다. [Package Explorer]보기에서 [SampleChap14.1]프로젝트를 마우스오른쪽단추로 찰칵하고 [Run]→[Run]을 선택한다. [Run]대화칸이 표시되는데 [Configuration]에 있는 [RuntimeWorkbench]를 선택한 상태에서 왼쪽밑에 있는 [New]를 찰칵한다. 그리고 이 대화칸의 제일 웃단에 있는 [Name]에 《HelloRcp》를 입력한다. 다음 [Run an application]칸을 선택하고 《sample.hello.HelloRcpApplication》을 선택한다(그림 14-9).

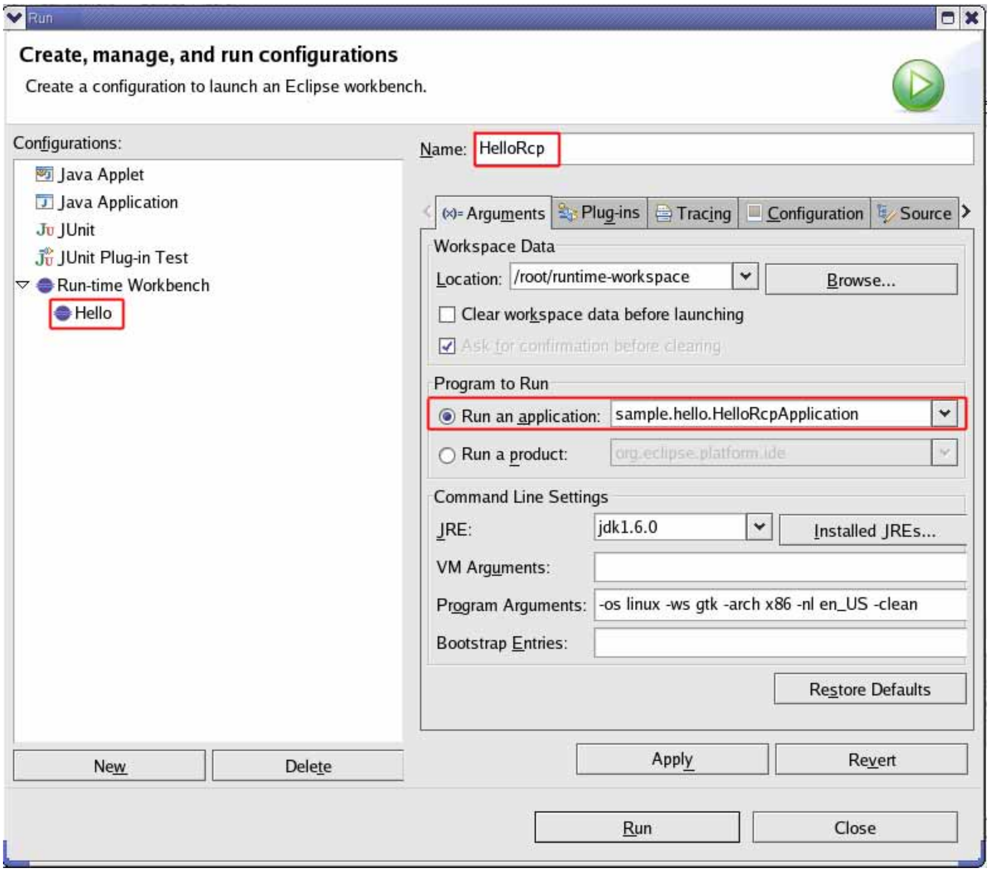


그림 14-9. RCP응용프로그람의 실행(1단계)

[Run]대화칸의 [Plug-ins]태브를 선택하고 [Choose plug-ins and fragments to launch from the list(기동하는 끼워넣기와 단편화실례를 목록에서 선택)]단추를 선택한다. 그러면 [Available plug-ins and fragments(사용가능한 끼워넣기 및 단편화)]안에 계층구조가 생긴다. 아래의 순서로 기동에 필요한 최소한의 끼워넣기를 선택한다(그림 14-10).

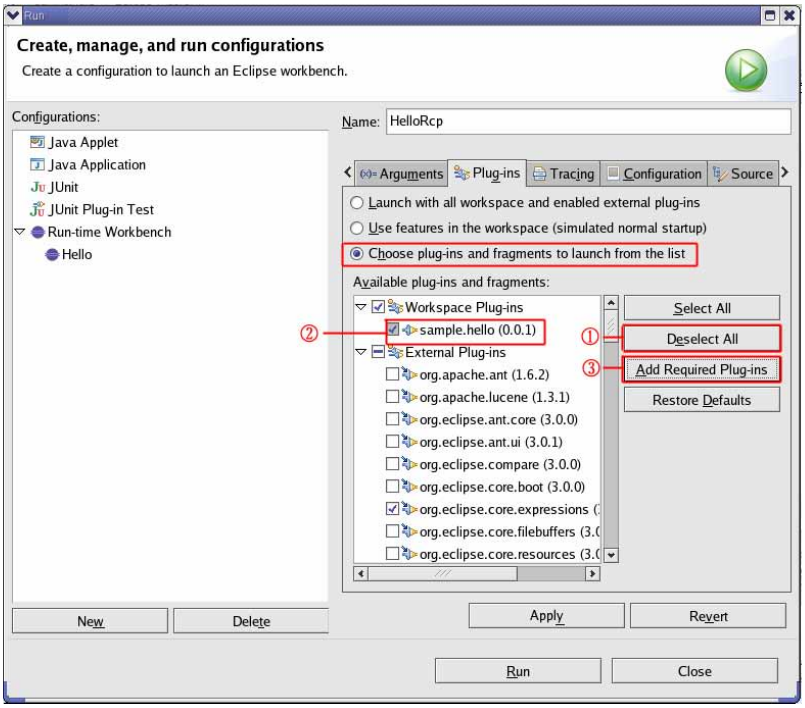


그림 14-10. RCP응용프로그람의 실행(2단계)

① 먼저 [Deselect All]를 찰칵한다.
② sample.hello(0.0.1)을 선택한다.
③ [Add required plug-ins]를 찰칵한다.

[Run]대화칸의 오른쪽 아래에 있는 [Run]를 찰칵하면 응용프로그람이 기동한다(그림 14-11). 이미 언급된것처럼 그 어떤 보기도 편집기도 없는 작업대가 기동한다.



그림 14-11. RCP응용프로그람의 실행

이제까지 RCP응용프로그람의 기초에 대하여 설명하였다. 다음은 본격적인 RCP응용프로그람인 RSS열람응용프로그람의 작성을 진행한다.

제3절. Eclipse RCP를 리용한 RSS열람응용프로그람

14.3.1. 견본응용프로그람의 개요

그러면 실제로 Eclipse RCP를 리용하여 견본응용프로그람을 시험해본다. RSS열람이라는것은 Web거점의 갱신상태나 개요가 서술되여있는 RSS파일을 Web로부터 얻어 여러가지 거점의 갱신상태를 검사하거나 관찰할수 있는 응용프로그람이다.

RSS열람응용프로그람을 가동시키려면 RSS파일을 Web에서 URL을 지정하여 얻어야 한다. 이 장에서는 처음에 견본응용프로그람의 개요를 설명한 후에 견본프로젝트를 반입한다.

다음은 견본응용프로그람을 실제로 실행하여 응용프로그람의 거동을 관찰한다. 정적인 원천코드가 어떠한 관계에 있는가를 설명한다.

이 견본응용프로그람에서는 JFace, SWT 등의 API를 많이 사용하고 있지만 Eclipse RCP의 해설에 기본을 두고 JFace, SWT 등에 대한 구체적인 설명은 략한다. 상세한 정보가 필요한 경우 Eclipse의 API 문헌, eclipse.org의 공식문헌 등을 참조한다.

14.3.2. 견본응용프로그람의 구성

SampleChap14.2프로젝트를 생성한다. 그림 14-12와 같은 프로젝트구성으로 되도록 하는것이 목표이다. 여기서는 기본적으로 설명하려는 부분에 대한 코드만을 서술하였다. 작성방법에 대하여는 이 책의 [3.1.3 프로젝트의 작성]에서 설명되였으므로 참조한다.

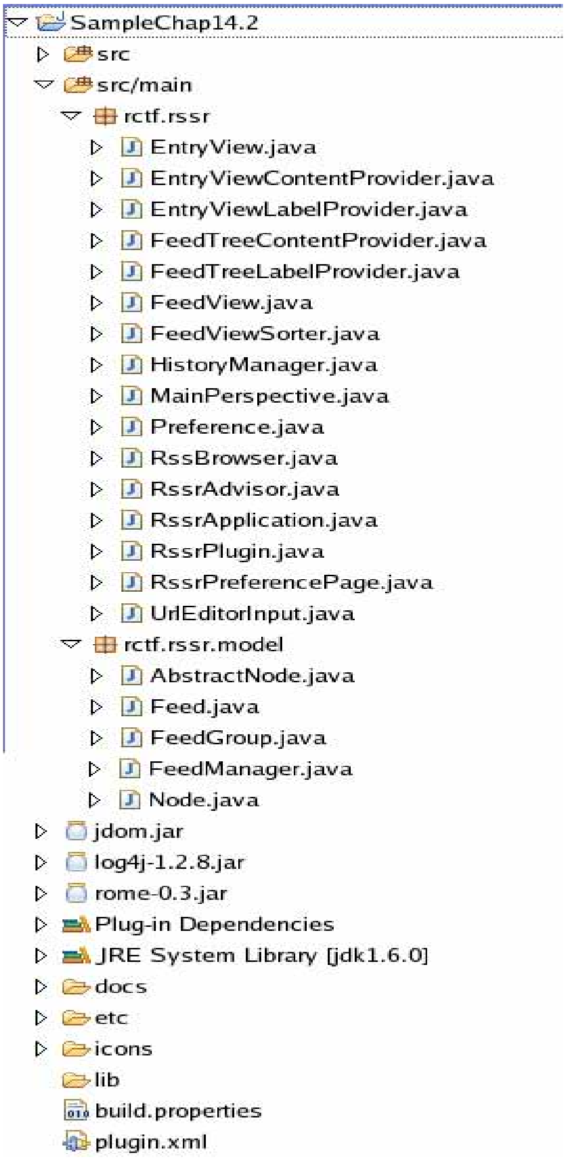


그림 14-12. 견본응용프로그람의 프로젝트구성

14.3.3. 견본응용프로그람의 실행

먼저 견본응용프로그람이 완성된 경우에 프로그람의 실행방법을 서술한다. [Package Explorer]보기의 《SampleChap14.2》프로젝트를 오른쪽단추로 찰칵하고 [Run]→[Run]을 선택한다. 그러면 [Run]대화칸이 나타난다. [Run]대화칸안의 [Configuration]에서 [Run-time Workbench]를 선택하고 왼쪽아래의 [New]를 찰칵한다. 그리고 [Run]대화칸의 제일상단에 있는 [Name]에 《RSSR》라고 기입한다. 마지막으로 [Run an application]단추를 선택하고 내리펼침목록으로부터 《rctf.rssr.RSSR》를 선택한다(그림 14-13).

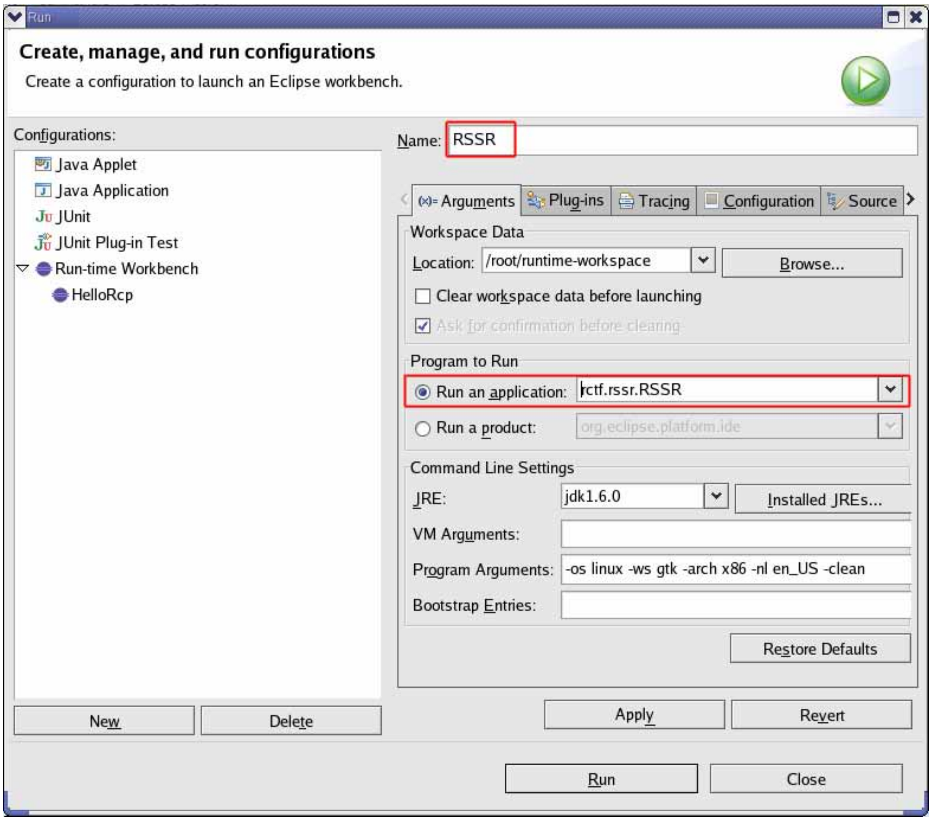


그림 14-13. 견본응용프로그람의 실행(1단계)

다음 [Run]대화칸의 [Plug-ins]태브를 선택하고 [Choose plug-ins and fragments to launch from the list]단추를 선택한다. 그러면 [Available plug-ins and fragments]안에 계층구조가 생긴다. 아래의 순서로 하면 기동에 필요한 최소한의 끼워넣기를 자동적으로 선택한다(그림 14-14).

① [Deselect All]를 찰칵한다.

② rctf.rssr(0.0.1)을 선택한다.

③ [Add required plug-ins]단추를 찰칵한다.

[Run]대화칸 오른쪽아래에 [Run]를 찰칵하면 응용프로그람이 기동한다.

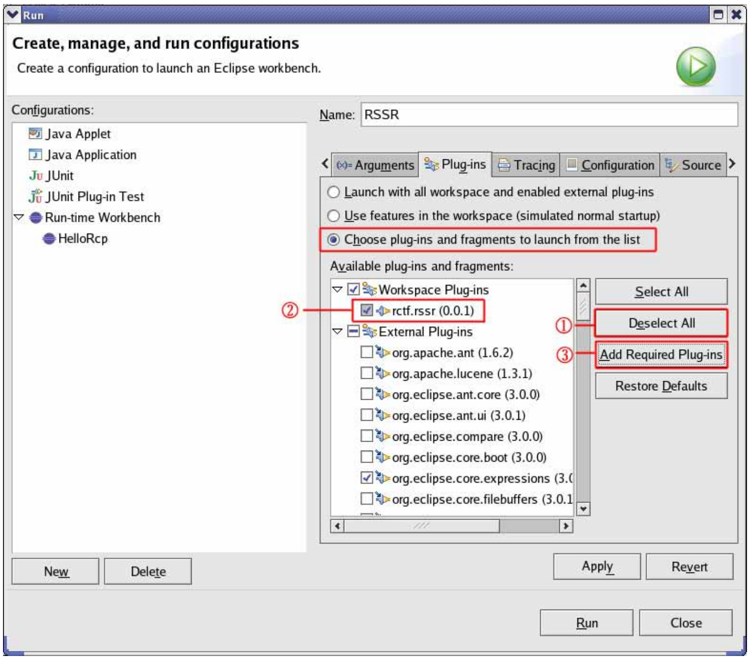


그림 14-14. 견본응용프로그람의 실행(2단계)

14.3.4. 견본응용프로그람의 사용방법

그림 14-15와 같은 화면이 나타나면 기동이 성공한것이다. 이 응용프로그람에는 RSS열람투시에서 차림표띠, [Feed]보기, [기사]보기가 구비되여있다.

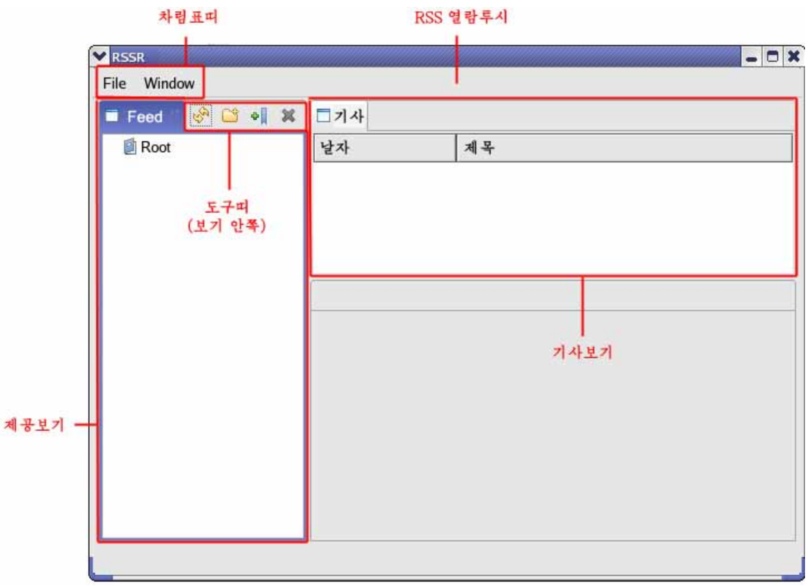


그림 14-15. 견본응용프로그람의 기동화면

먼저 RSS제공을 그룹화하기 위해 서류철을 추가한다. [Root]를 선택하고 [Feed]보기의 [New folder]추가단추를 누르면 [“Root”에 등록부추가]대화칸이 현시되는데 입력란에 《IT새소식》이라고 쓰고 [OK]를 찰칵한다(그림 14-16). 그러면 [Feed]보기에 [IT새소식]이라는 이름의 등록부아이콘이 추가된다.

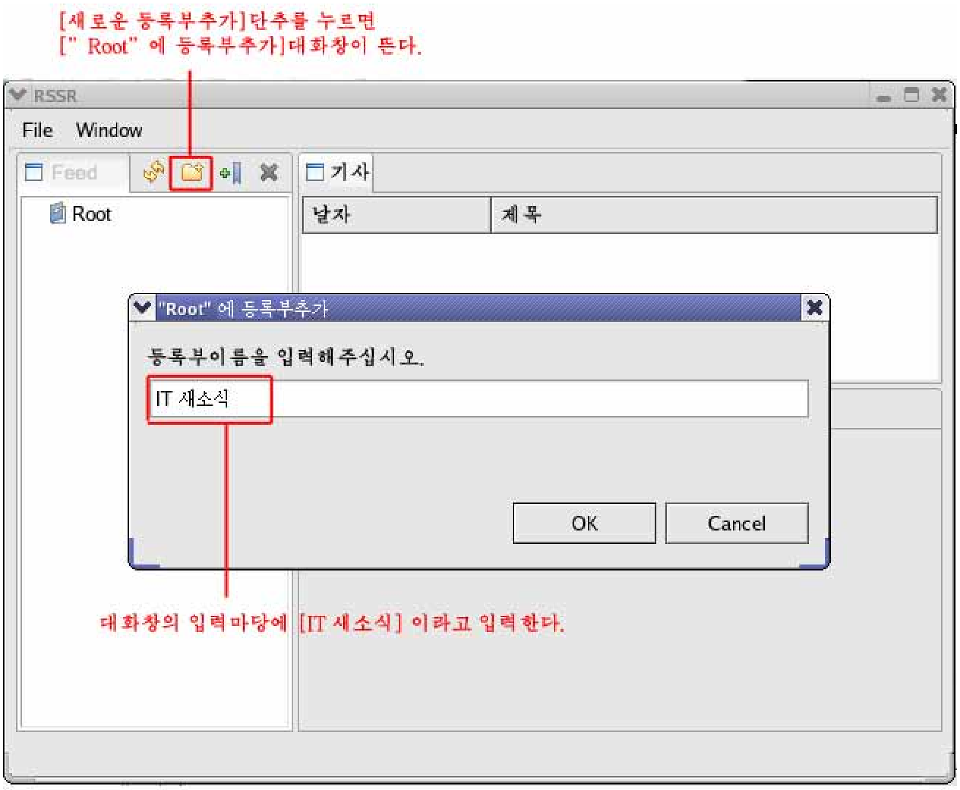


그림 14-16. 등록부의 추가방법

다음으로 실제로 RSS의 제공을 추가해본다. 이미 작성한 [IT새소식]등록부를 선택하고 [Feed]보기의 [New Feed]단추를 찰칵한다. [“IT새소식”에 제공추가]대화칸이 현시되는데 본문마당에 아래와 같은 URL을 입력한다(그림 14-17).

http://www.theserverside.com/rss/theserverside-rss2.xml

[OK]를 누르면 [Feed]보기에 새로운 아이콘과 RSS제공의 URL이 표시된다. URL을 두번 찰칵하면 [기사]보기에 제공되여있는 새소식의 기사일람이 현시된다. 기사일람으로부터 한개 기사를 두번 찰칵해본다. 자동적으로 Web열람기가 뜨고 방금 두번 찰칵한 기사내용이 서술되여있는 Web페지가 현시된다.

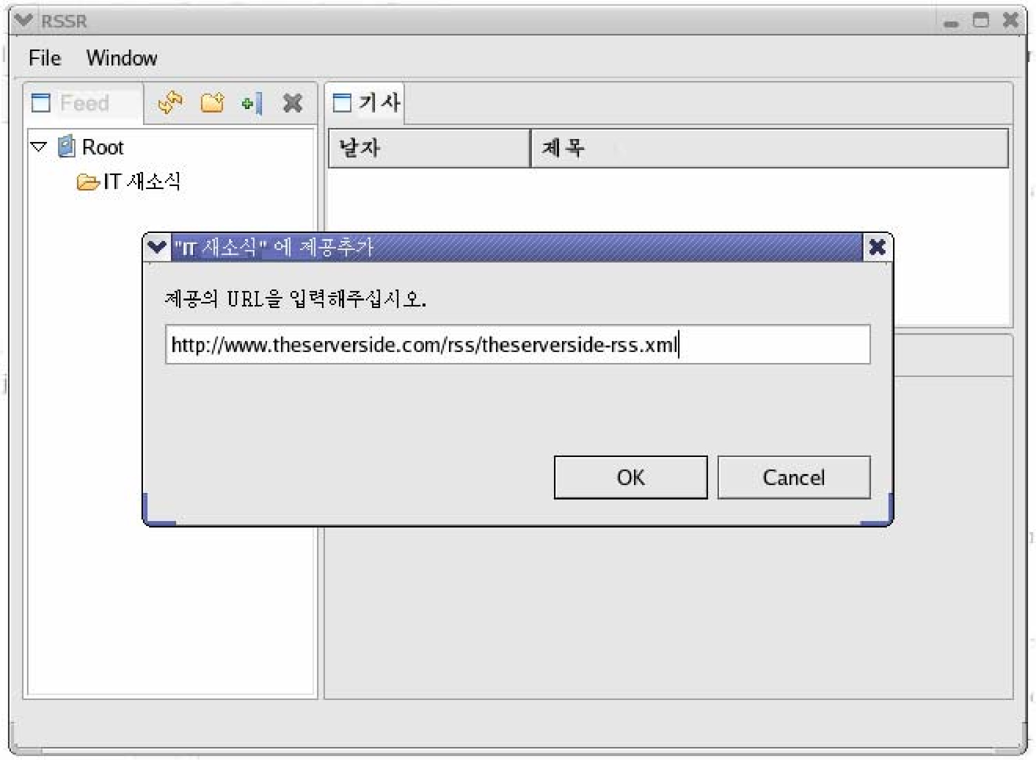


그림 14-17. RSS제공의 추가방법

제4절. 견본응용프로그람의 원천코드해설

Ecllipse RCP를 리용한 응용프로그람에서도 보통의 끼워넣기개발과 개발순서가 달라지지 않았다. 그러나 다음의 두가지 점에서는 Eclipse RCP의 응용프로그람이 보통의 끼워넣기개발과 차이난다.

- 반드시 기정투시를 정의하여야 한다.
- 추상클라스인 WorkbenchAdvisor를 계승한 클라스가 꼭 한개 필요하다.

14.4.1. plugin.xml

plugin.xml에 끼워넣기명시를 서술한다.

여기서는 기정의 Eclipse끼워넣기확장점에 대하여 확장을 정의한다.

앞에서 견본응용프로그람은 RSS열람투시우에 차림표띠, [Feed]보기, [기사]보기, Web열람기(SWT열람기)편집기가 구비된 구성을 하고있다고 했다. 어째서 Web열람기(SWT열람기)편집기라고 썼는지 리해하기 힘들수도 있다. 이 RCP응용프로그람에서는

Web열람기를 편집기로, 즉 Eclipse에서 편집기용 끼워넣기(org.eclipse.ui.editors)를 확장한 끼워넣기로 취급한다는것이다.

그러면 Web열람기(SWT열람기)의 의미는 무엇인가. 이번에 리용하는 열람기는 SWT열람기라고 부르는 Web열람기이다. 이 SWT열람기는 SWT에 정의되여있는 Browser클라스를 리용하고있다. 이 Browser클라스가 하는 일은 아주 간단한데 SWT에 들어있는 Win32 OLE(Object Linking and Embedding)를 호출하여 Internet Explorer의 화면을 SWT의 부분품우에 붙여놓는것이다. 기본적으로 사용자가 정의하는 투시는 org.eclipse.ui.perspective에 대한 확장, 보기는 org.eclipse.ui.views에 대한 확장, 편집기는 org.eclipse.ui.editors에 대한 확장으로 끼워넣기명시로 정의한다. 그리고 개개를 확장점에 정의된 대면을 실현하는 형식으로 클라스를 작성하고 끼워넣기명시에 서술한다. 목록 14-5를 본다. 이 목록에는 [Feed]보기의 끼워넣기명시를 서술하였다.

목록 14-5. plugin.xml에서 보기의 확장정의

```
<extension
        point="org.eclipse.ui.views">
    <view
        allowMultiple="false"
        class="rctf.rssr.FeedView"
        category="rctf.rssr"
        name="제공"
        id="rctf.rssr.FeedView"/>
```

extension요소의 point속성에 확장점을 서술하고 view요소안의 class속성에 확장점에서 정의된 대면을 실현한 클라스의 이름을 서술한다. 일반적으로 Id속성에는 보통 class속성과 같은 이름을 서술하는것이 좋다. allowMultiple속성으로는 보기를 여러개 펼치겠는가 아니면 하나만 펼치겠는가를 설정한다. false로 지정하면 투시안에 한개의 보기밖에 둘수 없다. category속성은 여러개의 보기를 묶어서 취급할 때 지정한다.

우의 plugin.xml의 내용은 plugin.xml용편집기로 편집할수 있다. 한번 plugin.xml의 내용이 어떻게 되여있는지 보아두는것이 좋다. Eclipse의 끼워넣기를 작성할 때는 그냥 본따는 기분으로 하는것도 큰것이다. 기존의 끼워넣기에서 비슷한 기능을 찾아 그 plugin.xml의 해당부분을 복사 및 붙여넣기하여 클라스의 실현도 흉내낼수 있다. 내용을 한번 보아두지 않으면 어떻게 plugin.xml의 내용을 복사 및 붙여넣기하면 좋은지도 알수 없게 되고말기때문에 한번 내용을 보아둔다. 목록 14-6은 필수끼워넣기의 정의이다. 이 끼워넣기에서 필요한 끼워넣기ID를 서술한다. 이 RCP응용프로그람에서는 3개의 끼워넣기

《org.eclipse.core.runtime》, 《org.eclipse.ui》, 《org.eclipse.jface》이 필수끼워넣기이다. Eclipse의 끼워넣기기동조수로부터가 아니라 독립형으로 Eclipse RCP응용프로그람을 기동하는 경우 다른 필요한 끼워넣기 (org.eclipse.update.configurator 등)가 있다는데 주의한다.

목록 14-6. 필수끼워넣기의 정의

```
<requires>
        <import plugin="org.eclipse.core.runtime"/>
        <import plugin="org.eclipse.ui"/>
        <import plugin="org.eclipse.jface"/>
</requires>
```

그림 14-18은 이번 RCP응용프로그람에서 사용하는 서고이다. 작성할 때 필요하므로 plugin.xml에 서술한다. 아래에서 이번 견본응용프로그람에서 사용하는 서고를 간단히 설명한다.

Log4J는 Jakarta프로젝트에서 작성한 로그를 위한 서고이다. 프로젝트 등에서 실제로 사용된적이 많다. 상세한 사용방법에 대하여는《http://logging.apache.org/log4j/docs/》를 참조한다.

JDOM은 XML의 해석을 쉽게 하기 위한 서고이다. 일반적으로 DOM, SAX의 실현인 Xerces를 리용하는것이 많다고 볼수 있지만 DOM, SAX는 쓰기 쉬운 API라고 말할수는 없다. JDOM은 그런 측면에서 DOM, SAX와 비교하면 아주 쉽게 다룰수 있는 API를 가진 서고라고 말할수 있다. JDOM의 상세한 내용은《http://www.jdom.org》를 참조한다.

Rome는 XML문서로 인터네트로부터 얻을수 있는 RSS제공을 간편하게 다루기 위한 서고이다. RSS제공을 다루기 위한 서고는 대단히 많지만 이번에는 비교적 가볍고 쓰기 쉬운 Rome을 선택하였다. 구체적인 사용방법은《http://wiki.java.net/bin/view/Javawsxml/Rome》을 참조한다.

plugin.xml의 서술에서 해설할 부분은 많지만 아래에서 원천코드와 관련이 있는 부분만을 추려서 설명한다.



그림 14-18. 끼워넣기에서 사용하는 서고

14.4.2. RssrApplication.java

Eclipse RCP응용프로그람을 작성하는데는 확장점에 있는 org.eclipse.core.runtime.applications에 대하여 끼워넣기명시에 org.eclipse.core.runtime.applications를 확장하는 등의 선언을 서술하고 (그림 14-19), 확장점에 정의되여있는 대면인 IPlatformRunnable을 실현하는 클라스를 정의할 필요가 있다. 실현하는 이 대면은 [Extension point descrition]의 [API Information]에 서술되여있다.

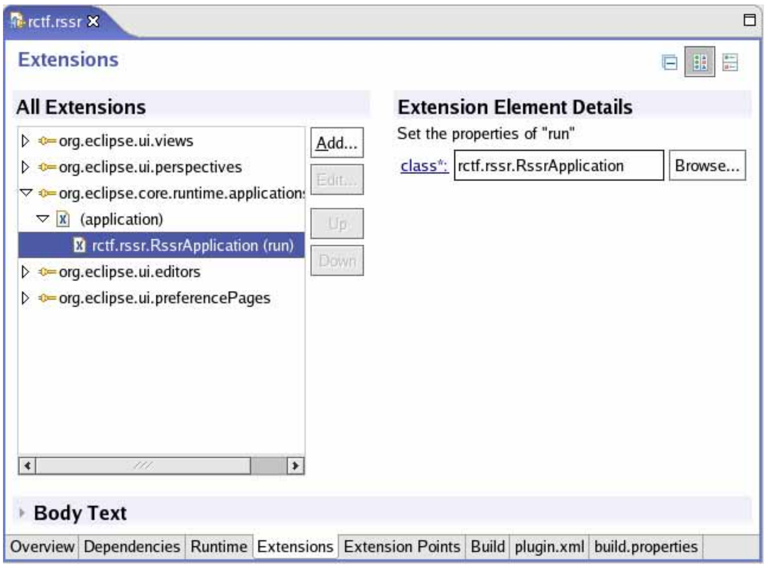


그림 14-19. 끼워넣기명시에 대하여 org.eclipse.core.runtime.applications의 확장선언

목록 14-7. RssrApplication.java의 class선언부

```
public class RssrApplication implements IplatformRunnable {
```

이 클래스에서 중요한것은 run메쏘드의 적용이다. 실제로 RCP응용프로그람이 기동되면 처음에 메쏘드가 호출된다. 이 run메쏘드에 동작내용을 서술한다.

목록 14-8. RssrApplication.java에서 run메쏘드의 적용

```
public Object run(Object args) throws Exception {
    initLog4j();
    WorkbenchAdvisor workbenchAdvisor = new RssrAdvisor();
    Display          display = PlatformUI.createDisplay();

    try {
        int returnCode= PlatformUI.createAndRunWorkbench
                          (display, workbenchAdvisor);

        if (returnCode == PlatformUI.RETURN_RESTART) {
            return IPlatformRunnable.EXIT_RESTART;
        }

        return IPlatformRunnable.EXIT_OK;
    } finally {
        if (display != null) {
        display.dispose();
        }
    }
}
```

run메쏘드에서는 처음에 log4j를 초기화하고 WorkbenchAdvisor의 대면과 SWT의 Display클라스의 대면을 생성한다. 이것을 리용하여 try구문안에서 실제로 작업대를 기동하며 귀환코드를 얻는다.

최종적으로 finally구문안에서 Display클라스의 대면이 처리되도록 하고있다.

14.4.3. RssrAdvisor.java

목록 14-9를 보면 알수 있듯이 RssrAdvisor클라스는 Eclipse RCP응용프로그람을 작성할 때 추상클라스인 WorkbenchAdvisor클라스를 계승하고있다. WorkbenchAdvisor클라스는 Eclipse RCP에서 가장 중요한 클라스이므로 WorkbenchAdvisor를 계승한 RssrAdvisor와 같이 해설한다.

목록 14-9. RssrAdvisor클라스의 선언부

```
public class RssrAdvisor extends WorkbenchAdvisor {
```

WorkbenchAdvisor클라스는 작업대, 작업대창문의 초기화도 하고 열기도 하는 등 수명에 관한 사건을 처리하는 메쏘드의 집합이라고 할수 있다.

그림 14-21, 표 14-1, 표 14-2, 표 14-3에서 기본적인 메쏘드를 참조한다.

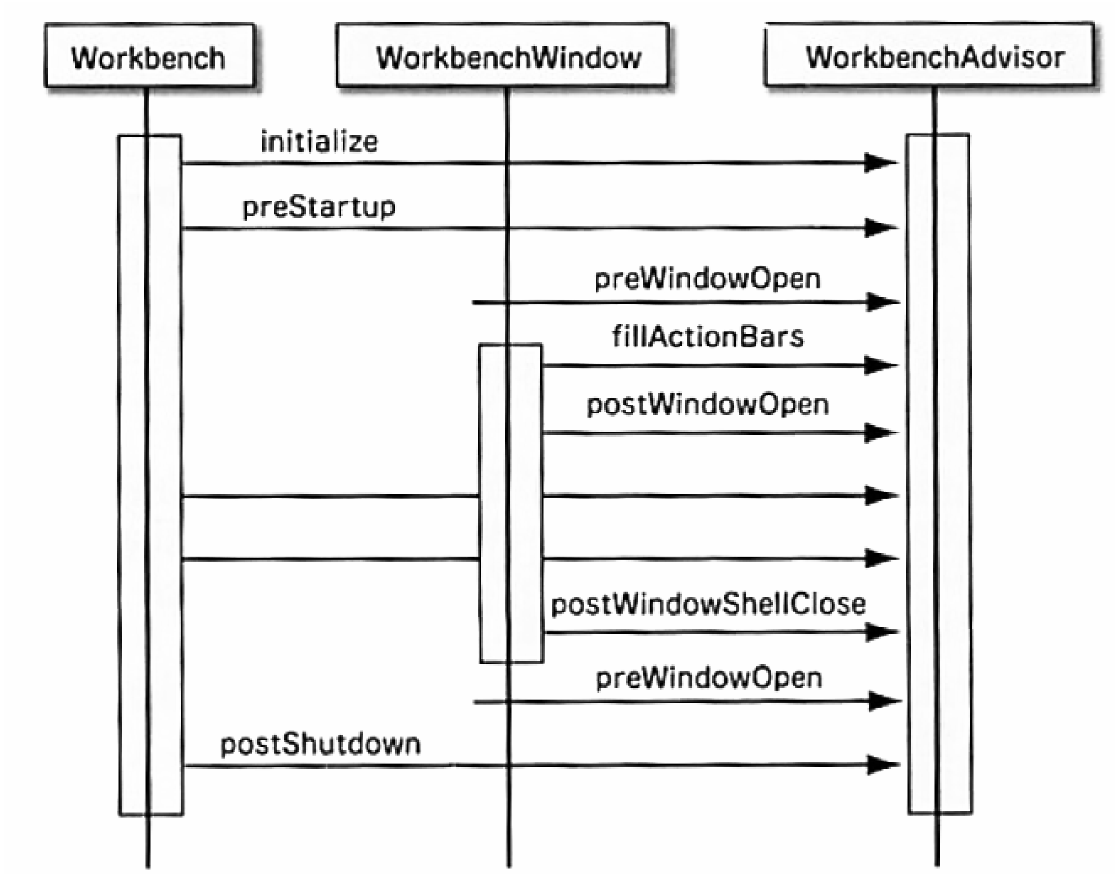


그림 14-20. 작업대의 수명에 관한 메쏘드

표 14-1. 작업대의 수명사건에 관한 갈구리(hook)메쏘드

메쏘드이름	설 명	인 수
Initialize	초기화처리용 메쏘드이다. 제일 처음에 호출된다. 지령행인수의 해석, 화상 및 아이콘의 준비와 같은 처리 등에 사용한다.	IWorkbenchConfigure
preStartup	초기화처리후 첫 창문이 열리기 전에 호출되는 메쏘드이다. 처음에 어느 편집기와 보기를 열겠는가를 결정하는 등의 처리에 사용한다.	
postStartup	모든 창문이 열린 다음 사건순환이 시작되기 전에 호출되는 메쏘드이다. 자동적으로 프로쎄스를 개시하는 처리나 암시문의 표시처리 등에 사용한다.	
preShutdown	사건순환이 완료된 다음 창문이 모두 닫기기 전에 호출되는 메쏘드이다.	
postShutdown	작업대종료시에 모든 창문이 닫긴 후에 호출되는 메쏘드이다. 현재의 응용프로그람의 상태를 보존하거나 초기화처리의 후처리 등에 사용한다.	

표 14-2. 작업대창문의 수명사건에 관한 갈구리메쏘드

메쏘드이름	설 명	인 수
preWindowOpen	WorkbenchWindow클라스의 구성자에서 호출되는 메쏘드이다. 창문이 차림표띠를 가지는가 안가지는가 하는 설정을 할 때 등에 사용한다.	IWorkbenchWindowConfigure
fillActionBars	preWindowOpen메쏘드의 완료 후에 인차 호출되는 메쏘드이다. 차림표띠나 도구띠의 코드작성을 하는 부분이다. 숙련될 때까지는 다루기가 어렵다.	IWorkbenchWindow IActionBarConfigureFlags
postWindowRestore	보존된 상태로부터 창문을 복귀시킬 때 사용되는 메쏘드이다. 복귀시키지 않을 때에는 사용하지 않는다.	Iworkbench WindowConfigure
postWindowCreate	창문이 보존된 상태에서 복귀한 직후나 새로 창문이 작성된 후에 호출되는 메쏘드이다. 그러나 이 메쏘드가 호출된 시점에서는 아직 창문이 열려 있지 않다는데 주의한다.	IWorkbenchWindowConfigure
openIntro	내부부품을 작성할 때 창문이 열린 직후에 호출되는 메쏘드이다.	IWorkbenchWindowConfigure
postWindowOpen	작업대창문이 열린 직후에 호출되는 메쏘드이다. 창문안의 화면부분품을 약간 변경시키는데 사용한다. 또한 목록으로 제목을 설정하고 창문의 크기를 설정하는 경우에 사용한다.	IworkbenchWindowConfigure

preWindow ShellClose	작업대창문(SWT의 Shell클라스의 대면)이 닫기기 전에 호출되는 메쏘드이다. 《정말로 창문을 닫겠습니까?》 등의 대화칸을 현시하려는 경우에 사용한다.	IWorkbenchWindow Configure
postWindow Close	작업대창문이 닫긴 후 호출되는 메쏘드이다. preWindowOpen에서 진행한 작업의 후처리 등에 사용한다.	IWorkbenchWindow Configure

표 14-3. 작업대의 사건순환에 관한 메쏘드

메쏘드이름	설 명	인 수
eventLoop Exception	사건순환안에서 예상치 않았던 례외가 발생되였을 때 호출되는 메쏘드이다. 일반적으로 로그기록을 진행한다.	Throwable
eventLoop Idle	사건순환에서 아무것도 하지 않는 경우 Display에서 호출되는 메쏘드이다.	Display

작업대는 보통 Eclipse를 기동했을 때 한번만 펼쳐진다. Eclipse를 기동했을 때 우리가 볼수 있는것은 작업대창문이다. 작업대창문은 차림표띠의 [Window]→[New Window]을 선택하여 마음대로 작성할수 있다. 그러나 작업대자체는 반드시 하나의 응용프로그람에 하나밖에 작성할수 없다. 종합적으로 말한다면 응용프로그람과 작업대의 관계는 1대1의 관계이지만 작업대와 작업대창문사이의 관계는 1대 다수의 관계이라고 할수 있다. 그러므로 표 14-2의 작업대창문의 수명사건에 관한 메쏘드들은 작업대창문을 띄운 회수만큼만 호출되는것으로 된다.

이제는 RssrAdvisor클라스의 원천코드를 작성해본다.

먼저 getInitialWindowPerspectiveId메쏘드이다. 이 메쏘드는 WorkbenchAdvisor클라스의 유일한 추상메쏘드이다. 최초의 작업대창문을 위한 투시의 ID를 돌려주기 위한 메쏘드이다.

이 메쏘드는 응용프로그람이 시작되면서 작업대가 작성될 때 호출된다. 목록 14-10은 RssrAdvisor클라스에서 메쏘드의 적용이다.

목록 14-10. getInitialWindowPerspectiveId메쏘드

```
public String getInitialWindowPerspectiveId() {
    return MainPerspective.class.getName();
}
```

견본응용프로그람에서는 클라스의 이름과 끼워넣기로부터 식별되는 ID가 동일하게 되여있으므로 이 견본응용프로그람의 유일한 투시인 MainPerspective클라스의 이름을 돌려 준다.

다음으로 작업대, 작업대창문의 수명사건에 관한 메쏘드를 차례로 해설한다.

RssrAdvisor클라스의 preStartup메쏘드와 preShutdown메쏘드는 우의 표에서 본바와 같이 작업대수명사건에 관한 메쏘드이다. 각각 최초의 창문이 열리기 전에 호출되는 메쏘드와 사건순환완료 후 창문이 모두 닫기기 전에 호출되는 메쏘드이다. 이 견본 응용프로그람을 시작하는데 따라 전후처리를 수행하고있다.

다음에 preWindowOpen메쏘드의 적용을 목록 14-11에 보여준다.

목록 14-11. preWindowOpen메쏘드

```
public void preWindowOpen
            (IWorkbenchWindowConfigurer configurer) {
  super.preWindowOpen(configurer);
  configurer.setInitialSize(new Point(
      Preference.WORKBENCH_WIDTH.getInt(),
      Preference.WORKBENCH_HEIGHT.getInt()));
  configurer.setTitle("RSSR");
  configurer.setShowCoolBar(false);
}
```

목록 14-12를 살펴본다. 여기서는 작업대창문을 열기 전에 640×480으로 창문크기를 설정한다. 이 창문의 너비 640과 높이 480자체의 수자는 Preference클라스에서 정의하고있다. 640, 480이라는 수자가 설정되여있으며 이 클라스에서는 [Feed]보기의 놓임새의 수값들도 설정되여 있다. 이렇게 응용프로그람에서 리용하는 상수는 Preference클라스 등 상수를 다루는 클라스를 만들어 관리하면 편리하다.

또한 이 작업대창문의 제목도 설정하고있다.

목록 14-13의 fillActionBars메쏘드는 preWindowOpen메쏘드완료 후 인차 호출되는 메쏘드이다. 이 메쏘드안에서 호출되고있는 비공개메쏘드인 fillMenuBar메쏘드로 차림표띠에 《File》차림표와 《Open》차림표를 만들고있다.

목록 14-12. Preference클라스에서 작업대창문의 설정부분

```
public class Preference {
 … …
 /** 작업대창문의 폭*/
   public static final Preference WORKBENCH_WIDTH
       = new Preference("WorkbenchWidth", "640");

/**작업대창문의 높이 */
```

```
        public static final Preference WORKBENCH_HEIGHT
            = new Preference("WorkbenchHeight", "480");
    … …
```

목록 14-13. fillActionBars메쏘드

```
    public void fillActionBars(IWorkbenchWindow window,
              IActionBarConfigurer configurer, int flags) {
        super.fillActionBars(window, configurer, flags);
        if ((flags & FILL_MENU_BAR) != 0) {
            fillMenuBar(window, configurer);
        }
    }
    /**차림표띠의 구축*/
    private void fillMenuBar(IWorkbenchWindow window,
                        IActionBarConfigurer configurer) {
       IMenuManager menuBar = configurer.getMenuManager();
       menuBar.add(createFileMenu(window));
       menuBar.add(createWindowMenu(window));
    }
```

목록 14-14의 postWindowOpen메쏘드는 작업대창문이 열린 직후에 호출되는 메쏘드이다. 여기서 [Feed]보기의 계층구조에서 RSS제공을 한개 선택하면 기사보기에 기사일람이 표시되도록 setupFeedsViewListener메쏘드를 호출하여 감시자를 등록한다.

목록 14-14. postWindowOpen메쏘드

```
    public void postWindowOpen
       (IWorkbenchWindowConfigurer configurer) {
        super.postWindowOpen(configurer);
        //계층구조에서 제공을 선택할 때마다 기사일람이 나오게 하는 감시자를 등록
        setupFeedsViewListener();
        //기사일람에서 기사를 선택할 때마다 편집기(SWT열람기)로 열리도록 감
          시자를 등록
        setupItemsViewListener();
    }
```

setupFeedsViewListener메쏘드를 보기로 한다. 여기서는 [Feed]보기로부터 JFace의 TreeViewer(계층구조를 표시하는 보기)를 얻고 계층구조의 어느 요소가 선택

되였는가를 알아내는 감시자를 등록한다(목록 14-15).

목록 14-15. setupFeedsViewListener메쏘드

```
/**
* 제공일람에서 제공을 선택하면
* 입구점이 입구점일람에 표시되도록 설정한다.
*/
private void setupFeedsViewListener() {
   TreeViewer treeViewer = RssrApplication.getFeedView().getViewer();
   treeViewer.addSelectionChangedListener
         (new ISelectionChangedListener() {
         public void selectionChanged(SelectionChangedEvent event) {
            StructuredSelection ss =
               (StructuredSelection) event.getSelection();
            RssrApplication.getItemView().getViewer().
               setInput(ss.getFirstElement());
         }
   });
}
```

setupItemsViewListener메쏘드에서는 [기사]보기에서 선택한 기사에로의 련결을 RSS제공용의 서고인 Rome으로부터 얻는다. 그리고 거기서 얻어진 Web페지에로의 련결을 리용하여 편집기 RssBrowser를 열기 위한 감시자를 등록한다.(목록 14-16)

목록 14-16. setupItemsViewListener메쏘드

```
private void setupItemsViewListener() {
   RssrApplication.getItemView().getViewer().addOpenListener(
     new IOpenListener() {
        public void open(OpenEvent event) {
           IStructuredSelection selected =
                 (IStructuredSelection) event.getSelection();
           final SyndEntryI entry = (SyndEntryI) selected.getFirstElement();
           try {
              IWorkbenchPage activePage =
                 PlatformUI.getWorkbench().getActiveWorkbenchWindow()
                    .getActivePage();
              activePage.openEditor(new UrlEditorInput(
                    entry.getTitle(), entry.getLink()),
                    RssBrowser.class.getName());
```

```
                    HistoryManager.getInstance().read(entry.getLink());
                    RssrApplication.getItemView().getViewer().refresh();
                } catch (Exception e) {
                    log.error(e, e);
                    return;
                }
            }
        });
    }
```

eventLoopException메쏘드는 창문처리의 기본순환내에서 처리되지 않은 례외가 발생할 때 호출되는 메쏘드이다. 여기서는 오유를 기록하게 한다(목록 14-17).

목록 14-17. eventLoopException메쏘드

```
public void eventLoopException(Throwable exception) {
    log.error("Unhandled event loop exception", exception);
}
```

14.4.4. MainPerspective.java

Eclipse RCP의 응용프로그람에서는 기정투시를 반드시 정의하여야 한다. 견본응용프로그람에서는 MainPerspective가 그에 해당된다.

MainPerspective와 관련된 plugin.xml의 끼워넣기명시(목록 14-18)를 본다.

목록 14-18. MainPerspective와 관련된 끼워넣기명시

```
<extension point="org.eclipse.ui.perspectives">
     <perspective
          class="rctf.rssr.MainPerspective"
          fixed="true"
          name="RSS열람"
          id="rctf.rssr.MainPerspective"/>
</extension>
```

이 MainPerspective는 확장점인 《org.eclipse.ui.perspectives》를 확장한다. 또한 중요한것은 perspective요소의 fixed속성이다. 이 속성에 참을 설정하면 투시의 놓임새가 고정되고 투시안에서 보기의 이동이나 닫기가 불가능해진다.

이번 견본응용프로그람에서도 [Feed]보기와 [기사]보기가 닫기면 프로그람적인 불

안정이 생기므로 fixed속성을 참으로 지정한다.

MainPerspective클라스의 원천코드를 본다(목록 14-19).

목록 14-19. MainPerspective클라스

```
public class MainPerspective implements IPerspectiveFactory {
   public void createInitialLayout(IPageLayout layout) {
      layout.setEditorAreaVisible(true);
      String editorArea = layout.getEditorArea();
      layout.addView(FeedView.class.getName(),
         IPageLayout.LEFT, Preference.FEED_VIEW_
         LAYOUT_RATIO.getFloat(), editorArea);
      layout.addView(EntryView.class.getName(), IPageLayout.TOP,
         Preference.ENTRY_VIEW_LAYOUT_RATIO.getFloat(), editorArea);
   }
}
```

투시를 확장할 때 확장점에 정의되여있는 대면인 IPerspectiveFactory를 실현한다. createInitialLayout메쏘드의 적용에서는 단순히 보기를 어떻게 배치하겠는가를 정의한다.

14.4.5. 보기관련클라스

이 프로그람에서는 2개의 보기 즉 [Feed]보기와 [기사]보기를 정의하고 사용하고있다. 목록 14-20에 보기와 관련한 plugin.xml안의 끼워넣기명시를 보여준다.

목록 14-20. 보기와 관련한 끼워넣기명시

```
<extension point="org.eclipse.ui.views">
      <view
            allowMultiple="false"
            class="rctf.rssr.FeedView"
            category="rctf.rssr"
            name="제공"
            id="rctf.rssr.FeedView"/>
      <view
            class="rctf.rssr.EntryView"
            category="rctf.rssr"
            name="기사"
            id="rctf.rssr.EntryView"/>

      <category
```

```
            name="RSSR"
            id="rctf.rssr"/>
    </extension>
```

보기는 확장점으로 《org.eclipse.ui.views》를 확장하여 작성한다. view요소안에서 보기를 정의한 클라스의 이름과 id를 지정한다.

Category요소에서 보기를 분류하는 범주 《RSSR》를 정의한다. 그 범주에 각 보기가 속하여있는데 view요소의 category속성에서 부류ID를 서술하여 지정한다. view속성의 allowMultiple속성은 여러개의 보기를 펼치겠는가 아니면 하나만 펼치겠는가를 결정하는 속성이다. 기정은 《false》로 되여있다.

[Feed]보기를 리용하여 《RSS제공을 그룹화하는 서류철》이나 《RSS제공자체》를 사용자가 조작한다. 그러나 실제로 RSS제공의 URL이나 RSS제공정보는 Feed클라스에서 관리하고 RSS제공을 그룹화하는 서류철정보는 FeedGroup클라스가 관리한다. 또한 Feed클라스와 FeedGroup클라스를 통일적으로 다루기 위하여 Node대면을 정의하고있다. 이러한 클라스군은 보통 령역객체나 모델객체라고 부른다.

실제로 정보를 관리하는 모델객체와 사용자의 조작접수, 겉보기를 제공해주는 [Feed]보기를 결합할 필요가 있다. 이 결합을 해주는것이 JFace의 추상클라스인 Viewer클라스를 계승한 어떤 하위클라스이다.

Viewer의 하위클라스에서 자주 리용되는것이 계층구조의 겉모양을 가진 TreeViewer, 일람형식의 겉모양을 제공하는 ListViewer, 그리고 표형식의 TableViewer이다. 이번 견본프로그람은 [Feed]보기에서는 TreeViewer, [기사]보기에서는 TableViewer를 사용한다.

TreeViewer는 ILabelProvider대면을 실현한 클라스와 ITreeContentProvider를 실현한 클라스를 같이 사용한다. [Feed]보기에서는 ITreeContentProvider , FeedTreeContentProvider클라스와 iLabelProvider대면을 실현한 FeedTreeLabelProvider를 TreeViewer와 같이 사용하고있다.

FeedTreeLabelProvider클라스자체는 모델객체(Feed클라스나 FeedGroup클라스 등)와 [Feed]보기의 계층구조에 있는 아이콘이나 본문문자렬을 붙여야 한다.

FeedTreeContentProvider는 TreeViewer에 대한 입력으로 되는 모델객체, 또는 모델객체의 집합과 TreeViewer자신이 내부에 가지는 목록구조의 대응을 제공해준다.

[Feed]보기에서는 FeedViewSorter클라스를 제공이나 제공그룹을 정돈할 때 사용하고있다. JFace의 정돈을 하고싶을 때에는 ViewerSorter클라스를 계승한 클라스를 정의하여 사용한다. 그것이 FeedViewSorter클라스이다.

그림 14-21에 모델객체와 TreeViewer와 관련된 클라스와의 관계를 보여준다.

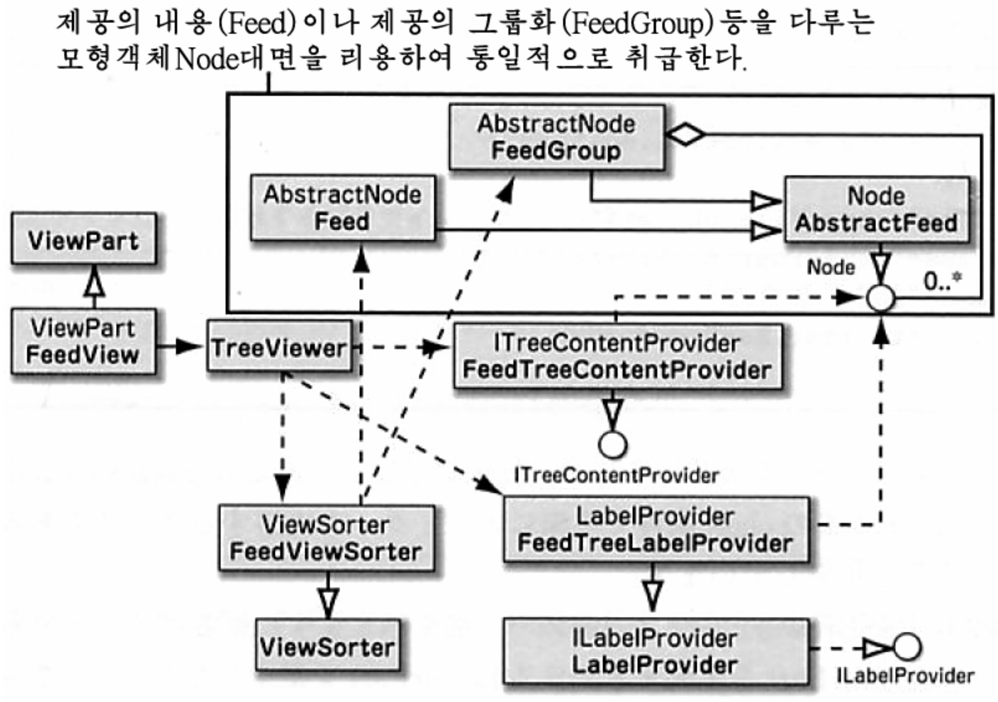


그림 14-21. FeedView클라스와 관련된 클라스무리

[기사]보기는 EntryView클라스에서 실현되여있지만 TreeViewer에서가 아니라 표형식의 걸모양을 제공하는 TableViewer를 사용하고있다. 이 TableViewer와 EntryViewContentProvider클라스, EntryViewLabelProvider클라스와 협동하여 기사일람을 표시한다.

EntryViewLabelProvider클라스는 ITableLabelProvider대면과 동시에 IFontProvider대면도 실현하고있다. 이렇게 하여 실제로 선택된 기사이외는 굵은 글자로 표시되고 선택된 기사는 보통의 얇은 글자로 표시되도록 하였다. 이장에서는 JFace를 구체적으로 소개하는것이 목적이 아니므로 이 정도로 설명을 한다.

14.4.6. 편집기관련클라스(RssBrowser클라스)

RssBrowser클라스는 [기사]보기에서 선택한 기사를 열람하기 위한 편집기를 실현한 클라스이다. 편집기와 관련된 끼워넣기명시를 본다(목록14-21).

목록 14-21. 편집기와 관련한 끼워넣기명시

```
<extension
        point="org.eclipse.ui.editors">
    <editor
        icon="icons/file_obj.gif"
        class="rctf.rssr.RssBrowser"
        name="RSS열람기"
        id="rctf.rssr.RssBrowser"/>
</extension>
```

편집기를 만들기 위하여 확장점인 org.eclipse.ui.editors를 확장하였다. editor요소의 class속성에서 실제로 편집기를 만드는 클라스인 RssBrowser클라스의 이름을 지정하고있다.

목록 14-22에 RssBrowser클라스 실현의 일부를 보여주었다. 확장점에서 정의된 대면 iEditorPart를 실현한 클라스 EditorPart를 계승하고있다. 이 클라스는 편집기라고 해도 SWT상에서 Window의 OLE를 리용하여 Internet Explorer를 표시하는 Browser클라스로 실현된다. 이 Browser클라스는 사용방법이 아주 간단하여 SWT의 화면부분품에 설정한후 Browser클라스의 대면에 대하여 setUrl(실제의 URL)을 지정하는것으로 쉽게 Web상의 페지를 현시해준다. Browser클라스는 간단히 사용할수 있기때문에 각종 Eclipse RCP응용프로그람의 교육용으로 사용되고있다.

목록 14-22. RssBrowser클라스(일부)

```
public class RssBrowser extends EditorPart {
    private Browser browser;
    public void createPartControl(Composite parent){
        browser = new Browser(parent, SWT.FILL);
        UrlEditorInput uei = (UrlEditorInput)getEditorInput();
        browser.setUrl(uei.getUrl());
    }
        … … (생략)… …
}
```

14.4.7. 설정대화칸관련클라스

그러면 마지막으로 설정대화칸에 대하여 보기로 한다. 이 설정대화칸은 일반적으로 차림표띠의 [Window]→[Preference]을 선택하여 나오는 페지이다. 이 책에서도 이 설정대화칸을 리용하여 Eclipse의 여러가지 설정을 진행하였다. 이번 견본프로그람에서

는 설정대화칸에서 설정을 한 대로 작성한다. 우선 끼워넣기명시를 본다(목록 14-23). 설정대화칸은 확장점 《org.eclipse.ui.preferencePages》에 대하여 확장을 진행하는 것으로 작성한다. page요소의 클라스속성에 실제로 설정대화칸을 실현하는 클라스의 이름을 서술한다.

목록 14-23. 설정대화칸과 관련한 끼워넣기명시

```
<extension
        point="org.eclipse.ui.preferencePages">
    <page
        class="rctf.rssr.RssrPreferencePage"
        name="RSSR설정"
        id="rctf.rssr.RssrPreferencePage"/>
</extension>
```

목록 14-24에서 이 견본프로그람의 설정대화칸을 실현하는 RssPreferencePage클라스를 보여주었다.

목록 14-24. RssPreferencePage

```
public class RssrPreferencePage extends
    FieldEditorPreferencePage implements IWorkbenchPreferencePage {
  public RssrPreferencePage() {
    super(FieldEditorPreferencePage.FLAT);
    setPreferenceStore(RssrPlugin.getInstance().getPreferenceStore();
  }
  public void init(IWorkbench workbench) {}
  protected void createFieldEditors() {
    addField(new FileFieldEditor(Preference.LOG4J_CONFIGURATION.key(),
        "label:log4jp", getFieldEditorParent()));
    addField(New DirectoryFieldEditor(Preference.RSSR_DIR.key(),
        "label:rssrdir", getFieldEditorParent()));
  }
}
```

구성자에서 setPreferenceStore메쏘드를 사용함으로써 자동적으로 설정내용을 Eclipse를 기동하고있는 콤퓨터에 보존하고있다. 또다시 기동할 때에는 자동적으로 이 설정내용을 읽어낸다. 이렇게 PreferenceStore는 설정내용을 보존할 때에 아주 편리하게 사용할수 있다.

FieldEditor는 설정대화칸안에서 하나의 설정을 진행하기 위한 마당을 정의한다. createFieldEditors메쏘드에서는 실제로 설정파일의 항목(label:log4jp, label:rssrdir)을 등록한다.

제5절. 견본응용프로그람의 독립형기동

지금까지는 Eclipse우에서 실행대화칸을 리용하여 기동하였지만 Eclipse RCP는 최소로 필요한 끼워넣기와 Eclipse부속의 startup.jar가 있으면 단독으로 기동할수 있다.

그러면 우선 독립형기동을 위한 순서에 대하여 설명한다.

사실 Eclipse3.0.1단계에서는 Eclipse RCP용의 반출기능이 아직까지 존재하지 않는다. 그러므로 SampleChap14.2를 끼워넣기와 같이 배포용으로 반출하여 필요한 끼워넣기를 Eclipse의 설치등록부안에 있는 Plugins등록부로부터 수동으로 복사한다. 물론 Eclipse의 설치등록부안에 있는 startup.jar도 수동으로 복사한다.

앞으로 Eclipse에서의 이러한 수동작업을 편리하게 하는 Eclipse RCP용 배포조수도 등장할것이다. 사실상 이 책을 집필하는 과정에도 여러번 부주의로 독립형기동이 실패하였다.

14.5.1. SampleChap14.2의 반출

처음에 SampleChap14.2프로젝트를 Eclipse용끼워넣기로 반출한다. 그러나 우선 끼워넣기로 동작시키는데 필요한 프로젝트안의 등록부를 선택하여야 한다. [Package Explorer]안의 SampleChap14.2프로젝트에 있는 plugin.xml을 두번찰칵하여 열고 [Build]페지를 연다. 왼쪽아래에 [Binary Build]라는 부분이 있는데 3개의 등록부 etc, icons, lib와 plugin.xml의 검사칸을 선택한다(그림 14-22). 이 조작을 하면 끼워넣기로 반출할 때에 우에서 지정한 등록부가 포함되는것으로 된다. 반대로 이 조작을 하지 않으면 필요한 서고나 아이콘파일이 포함되지 않아 프로그람이 기동하지 못하므로 주의해야 한다.

[Package Explorer]에서 SampleChap14.2프로젝트를 마우스오른쪽단추로 찰칵하고 [Export]을 선택하면 [Export]조수가 기동한다. [Export]조수에서 [Export Plug-ins and Fragments]대화칸의 [Avaliable Plug-ins and Fragments]가 표시되는데 그림 14-23와 같이 [Export Options]에서 [a directory structure]를 선택하고 [Directory]에 [/kut/eclipse/chap14/SampleChap14.2]을 지정한다.

이렇게 [/kut/eclipse/chap14/SampleChap14.2]이하의 등록부가 작성되고 그안에 끼워넣기가 복사된다. [Finish]를 1찰칵하면 반출이 진행된다.

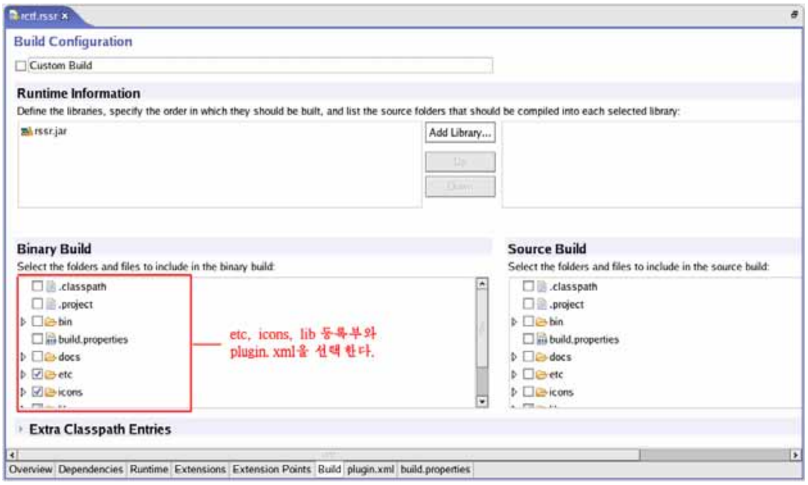


그림 14-22. 등록부와 파일의 선택대화칸

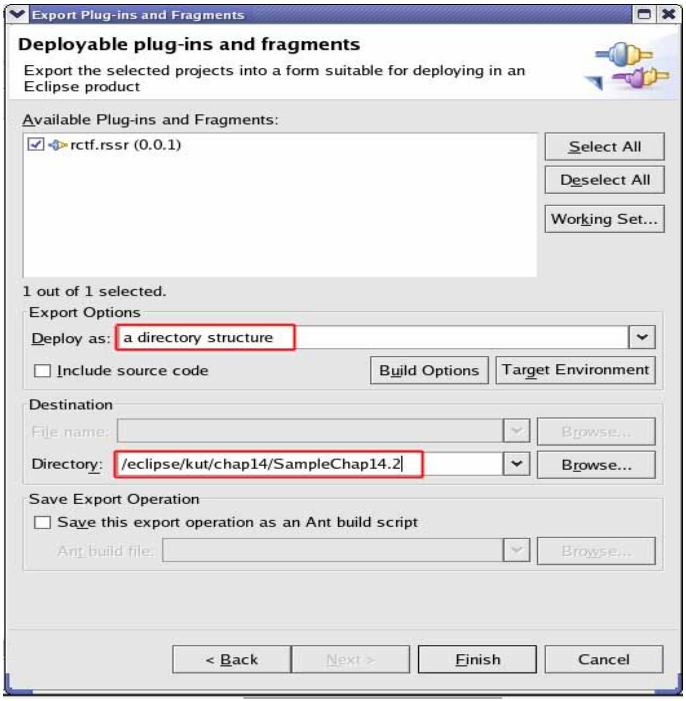


그림 14-23. 끼워넣기의 반출조수

14.5.2. startRssr.cmd의 작성

다음은 견본프로그람의 독립형기동에 필요한 파일 startRssr.cmd를 준비한다. 목록 14-25와 같은 파일을 작성하고 [/kut/eclipse/chap14/SampleChap14.2]등록부밑에 복사한다.

목록 14-25. startRssr.cmd

```
echo on
setlocal
cd %~dp0
javaw   cp startup.jar org.eclipse.core.launcher.Main   application rc
        tf.rssr.RSSR %*
endlocal
```

14.5.3. 기동에 필요한 끼워넣기를 수동으로 복사

그림 14-24에서 기동에 필요한 파일구성에 대하여 보여준다. 그림으로부터 알수 있는바와 같이 Eclipse설치등록부의 plugins등록부안에서 부터 필수끼워넣기등록부를 [/kut/eclipse/chap14/SampleChap14.2/plugins]등록부에 복사한다. 여기서 주의하지 않으면 안되는것이 org.eclipse.update.configurator끼워넣기이다. 이 끼워넣기는 Eclipse로부터 기동할 때에는 지정하지 않아도 되지만 독립형기동할 때에는 plugins등록부에 있는 끼워넣기의 plugin.xml을 읽어낼 때 반드시 필요한것이므로 복사한다.

마지막으로 Eclipse설치등록부에 있는 startup.jar를[/kut/eclipse/chap14/SampleChap14.2]등록부아래에 복사한 다음 준비를 완료한다.

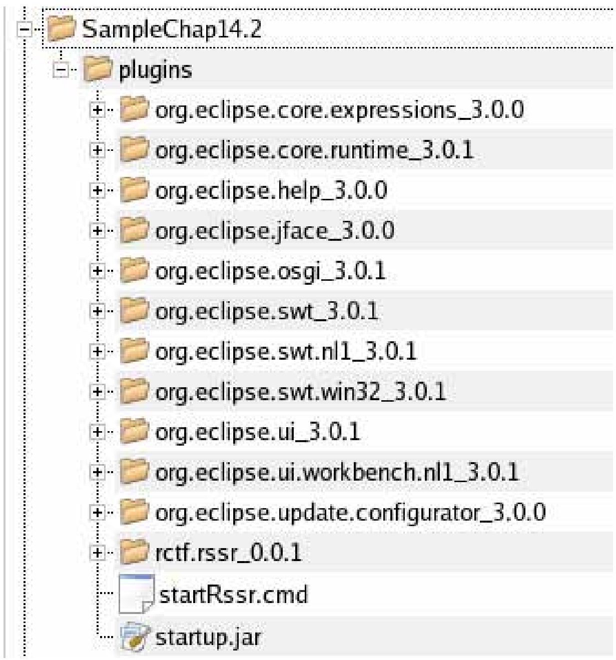


그림 14-24. 독립형기동할 때에 등록부구조

startRssr.cmd를 두번 찰칵하여 기동시켜 본다. 기동하지 않는 경우는 [/kut/eclipse/chap14/SampleChap14.2/configuration]등록부안에 기록파일이 작성되므로 참조한 다음 정확히 기동하도록 수정한다.

찾아보기

통합개발환경 Eclipse

집 필	김수홍, 김수현 김원규, 백남진	심 사	강 철
편 집	차현옥	교 정	유현순
장 정	서경애	콤퓨터편성	여은정

낸 곳	교육성 교육정보쎈터	인쇄소	교육성 교육정보쎈터
인 쇄	주체 97(2008)년 8월 10일	발 행	주체 97(2008)년 8월 20일

교-07-1308